

Lecture Notes in Computer Science

2889

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer

Berlin

Heidelberg

New York

Hong Kong

London

Milan

Paris

Tokyo

Robert Meersman Zahir Tari et al. (Eds.)

On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops

OTM Confederated International Workshops
HCI-SWWA, IPW, JTRES, WORM, WMS, and WRSM 2003
Catania, Sicily, Italy, November 3-7, 2003
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Robert Meersman
STAR Lab, Vrije Universiteit Brussel
Pleinlaan 2, Gebouw G-10, 1050 Brussels, Belgium
E-mail: Robert.Meersman@vub.ac.be

Zahir Tari
RMIT University
School of Computer Scienc and Information Technology
GPO Box 2476V, Melbourne, Victoria 3001, Australia
E-mail: zahirt@cs.rmit.edu.au

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

CR Subject Classification (1998): H.2, H.3, H.4, C.2, H.5, I.2, D.2.12, K.4

ISSN 0302-9743

ISBN 3-540-20494-6 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media
springeronline.com

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 10970584 06/3142 5 4 3 2 1 0

OTM 2003 General Co-chair's Message

We as General Chairs are rather proud to announce that the conference series we started in Irvine last year proved to be a concept that continues to attract a representative selection of today's research in distributed, heterogeneous yet collaborative systems, of which the Internet and the WWW are its prime examples.

Indeed, as large, complex and networked intelligent information systems become the focus and norm for computing, software issues as well as methodological and theoretical issues covering a wide range of topics, such as data and Web semantics, distributed objects, Web services, databases, workflow, cooperation, ubiquity, interoperability, and mobility for the development of Internet- and Intranet-based systems in organizations and for e-business, need to be addressed in a fundamental way. The second, 2003 edition of the "On The Move" (OTM) federated conference event provided an opportunity for researchers and practitioners to understand and publish these developments within their respective as well as within their broader contexts. It co-located the three related, complementary and successful conference series: DOA (Distributed Objects and Applications), covering the relevant infrastructure-enabling technologies, OD-BASE (Ontologies, DataBases and Applications of SEmanantics) covering Web semantics, XML databases and ontologies, and CoopIS (Cooperative Information Systems) covering the application of these technologies in an enterprise context through, for example, workflow systems and knowledge management. Each of these three conferences treated its topics within a framework of theory, conceptual design and development, and applications, in particular case studies and industrial solutions.

In 2003 we also invited a number of workshop proposals to complement the more archival nature of the main conferences with research results in a number of selected and more avant garde areas related to the general topic of distributed computing. For instance, the so-called Semantic Web has given rise to several novel research areas combining linguistics, information systems technology, and artificial intelligence, such as the modeling of (legal) regulatory systems and the ubiquitous nature of their usage. One such event was continued from last year, a so-called "Industry Program" workshop soliciting relevant case studies and best practice results from industry in the areas covered by On The Move 2003.

All three conferences and the associated workshops therefore shared the distributed aspects of modern computing systems, and the resulting application pull created by the Internet and the so-called Semantic Web. For DOA 2003, the primary emphasis stayed on the distributed object infrastructure; for OD-BASE 2003, it became the knowledge bases and methods required for enabling the use of formal semantics; and for CoopIS 2003, the main topic was the interaction of such technologies and methods with management issues, such as occur in networked organizations. These subject areas naturally overlap and many sub-

missions in fact also treat an envisaged mutual impact among them. As for the 2002 edition in Irvine, the organizers wanted to stimulate this cross-pollination with a program of shared famous keynote speakers (this year we got Sycara, Goble, Soley and Mylopoulos!), and encouraged multiple attendance by providing authors with free access to another conference or workshop of their choice.

We received an even larger number of submissions than last year for the three conferences (360 in total) and the workshops (170 in total). Not only can we therefore again claim a measurable success in attracting a representative volume of scientific papers, but such a harvest allowed the program committees of course to compose a high-quality cross-section of worldwide research in the areas covered. In spite of the increased number of submissions, the Program Chairs of the three main conferences decided to accept only approximately the same number of papers for presentation and publication as in 2002 (i.e., around 1 paper out of every 4–5 submitted). For the workshops, the acceptance rate was about 1 in 2. Also for this reason, we decided to separate the proceedings into two volumes with their own titles, and we are grateful to Springer-Verlag for their collaboration in producing these two books. The reviewing process by the respective program committees was very professional and each paper in the main conferences was reviewed by at least three referees. The On The Move federated event organizers therefore also decided to make both (sizeable) books of proceedings available to all participants of the conferences and workshops. Even though this meant that participants had extra weight to carry home.

The General Chairs are especially grateful to all the many people directly or indirectly involved in the setup of these federated conferences, and who in so doing made then a success. In particular we thank our eight conference PC co-chairs (DOA 2003, Bernd Krämer, Maarten van Steen, and Steve Vinoski; ODBASE 2003, Roger (Buzz) King, Maria Orlowska, and Rudi Studer; CoopIS 2003, Elisa Bertino and Dennis McLeod) and our 13 workshop PC co-chairs (Angelo Corsaro, Corrado Santoro, Priya Narasimhan, Ron Cytron, Ernesto Damiani, Brian Blake, Giacomo Cabri, Mustafa Jarrar, Anne Salaun, Elizabeth Chang, William Gardner, Tharam Dillon, and Michael Brodie), our publicity chair (Guillaume Pierre) and our publication chair (Kwong Lai), who together with their many PC members did a superb and professional job in selecting the best papers from the large harvest of submissions.

We do hope that again the results of this federated scientific event may contribute to your work and that we may see you all again, as well as many others, for next year's edition!

August 2003

Robert Meersman, Vrije Universiteit Brussel, Belgium
 Zahir Tari, RMIT University, Australia
 Douglas Schmidt, University of California at Irvine, USA

Organizing Committee

The OTM (On The Move) 2003 Federated Workshops involved a series of workshops co-located with the OTM 2003 Federated Conferences (namely CoopIS, DOA and ODBASE). These workshops were: Workshop on Human Computer Interface for Semantic Web and Web Applications (HCI-SWWA); Industry Program Workshop (IPW); Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES); Workshop on Regulatory Ontologies and the Modeling of Complaint Regulations (WORM CoRe); Workshop on Metadata for Security (WMS); and Workshop on Reliable and Secure Middlewares (WRSM).

The OTM 2003 Federated Workshops were proudly supported by the following institutions: RMIT University (School of Computer Science and Information Technology, Australia), Vrije University of Brussels (Department of Computer Science, Belgium), and University of Catania (Italy).

Executive Committee

Workshop General Co-chairs	Robert Meersman (Vrije U. of Brussels, Belgium) and Zahir Tari (RMIT U., Australia)
HCI-SWWA 2003 PC Co-chairs	Elizabeth Chang (Curtin U. of Technology, Australia) and Tharam Dillon (U. of Technology, Sydney, Australia)
IPW 2003 PC Co-chairs	Michael Brodie (Verizon Information Technology, USA) and Robert Meersman (VU Brussels, Belgium)
JTRES 2003 PC Co-chairs:	Angelo Corsaro (Washington U., USA), Ron Cytron (Washington U., USA) and Corrado Santoro (U. of Catania, Italy)
WORM 2003 CoRe PC Co-chairs	Mustafa Jarrar (Vrije U. of Brussels, Belgium) and Anne Salaun (e-consult, Belgium)
WMS 2003 PC Chair	Ernesto Damiani (U. di Milano, Italy)
WRSM 2003 PC Co-chairs	Brian Blake (Georgetown U., USA), Giacomo Cabri (U. of Modena, Italy) and Priya Narasimhan (Carnegie Mellon U., USA)
Publication Chair	Kwong Yuen Lai (RMIT U., Australia)
Local Organizing Chair	Corrado Santoro (U. of Catania, Italy)
Publicity Chair	Guillaume Pierre (Vrije U. of Amsterdam, The Netherlands)

HCI-SWWA 2003 Program Committee

P. Janecek	M. Urban	M.L. Bernard
E.G. Barriocanal	J. Yang	
T. Catarci	S. Simoff	
O. Pastor	J.A. Gutierrez	

IPW 2003 Program Committee

R. Agrawal	D. Jacobs	A. Servida
J. Angele	R. Jain	D. Sharp
S. Baker	V. Krishnamurthy	R.M. Soley
C. Bartolini	A. Léger	S. Tai
C. Bussler	F. Leymann	S.R. Thatte
M. Carey	A. Malhotra	J. Thomas
R. Cencioni	M. Marabello	B. Vatant
W. Ceusters	J. Miller	L. Vervenne
E. Cobb	J. Müller	S. Vinoski
F. Danza	A. Nori	A. Watson
J. Davies	A. Persidis	A. Zwegers
U. Dayal	F. Pititto	
U. Deppisch	M. Rusinkiewicz	
P. Hallock	P. Scoggins	

JTRES 2003 Program Committee

G. Bollella	D. Lea	J. Regehr
J. Cross	D. Locke	M. Rinard
P. Dibble	J. Loyall	D. Sharp
C. Gill	M.T. Higuera	D. Schmidt
G. Di Giore	D. Holmes	S. Vigna
M. De Miguel	D. Niehaus	J. Vitek
M. Fargetta	K. Nielsen	A. Wellings
M. Valls	A. Niessner	
D. Jensen	A. Poggi	

WORM CoRe 2003 Program Committee

A. Gangemi	M. Sintek	Y. Sure
B. Spencer	R. Meersman	Y. Pouillet
G. Wagner	R. Temmerman	
J. Breuker	S. Tabet	
M.F. Lopez	S. Louveaux	

WMS 2003 Program Committee

E. Chang	W. Jonker
S. De Capitani di Vimercati	M. Kudo
T. Dillon	P. Samarati
E. Fernandez-Medina	R. Tolksdorf
C. Farkas	X. Wang
C. Geuer-Pollmann	D. Wijesekera
P. Hallam-Baker	

WRSM 2003 Program Committee

D. Bakken	P. Ezhilchelvan
I. Rouvellou	R. Baldoni
J. Fabre	R. Friedman
N.F. Neves	

Table of Contents

Posters of the 2003 DOA (Distributed Objects and Applications) International Conference

On Enhancing Component-Based Middleware with Transactions	1
<i>Marek Prochazka, Romain Rouvoy, Thierry Coupaye</i>	
Online Consistency Checking for Replicated Objects	3
<i>Giovanni Masarin, Alberto Bartoli, Vance Maverick</i>	
Integrating Container Services with Pluggable System Extensions	5
<i>Andreas Leicher, Alexander Bilke, Felix Bübl, E. Ulrich Kriegel</i>	
SIG-SAT Trillo: A Full Scope Simulator for Nuclear Power Plants	7
<i>Manuel Díaz, Daniel Garrido</i>	
MobJeX: A Declaratively Configurable Java Based Framework for Resource Aware Object Mobility	11
<i>Caspar Ryan, Stephen Perry</i>	

Posters of the 2003 CoopIS (Cooperative Information Systems) International Conference

Trading Freshness for Performance in a Cluster of Replicated Databases	14
<i>Cécile Le Pape, Stéphane Gançarski, Patrick Valduriez</i>	
Regarding the Integration of Collaborative Applications into Virtual Worlds	16
<i>Roberta Lima Gomes, Guillermo J. Hoyos-Rivera, Jean Pierre Courtiat</i>	
Workflow Reconfiguration Using Chains	18
<i>Avigdor Gal, Eugeny Michailovsky, Mati Golani</i>	

Posters of the 2003 ODBASE (Ontologies, Databases and Applications of Semantics) International Conference

Modeling Semistructured Data by Using Graph-Based Constraints	20
<i>Ernesto Damiani, Barbara Oliboni, Elisa Quintarelli, Letizia Tanca</i>	
A Graph-Based Model for Semistructured Temporal Data	22
<i>Carlo Combi, Barbara Oliboni, Elisa Quintarelli</i>	

A Formal Approach to Knowledge Base Comparison and Interoperability	24
<i>Dan Corbett</i>	
COVA: A Web-Based Learning System for Content-Based Lecture Retrieval	26
<i>Guang-Ho Cha</i>	
An Ontology for Semantic Middleware: Extending DAML-S Beyond Web-Services	28
<i>Daniel Oberle, Marta Sabou, Debbie Richards, Raphael Volz</i>	
A Formalism of Contextualized Predicate Calculus for Information Integration	30
<i>Jijuan Zheng</i>	
An Ontology Based Visual Tool for Query Formulation Support	32
<i>Tiziana Catarci, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, Sergio Tessaris</i>	
Industry Program Workshop (IPW)	
Workshop PC Chairs' Message	34
<i>Robert Meersman, Michael Brodie</i>	
Revaluation of a Large-Scale Thesaurus for Multi-media Indexing: An Experience Report	35
<i>Dirk Deridder, Peter Soetens</i>	
On Storing and Querying XML Data within Network-Enabled Embedded Systems	46
<i>Hyoseop Shin</i>	
An Object-Oriented Invocation Layer for the Java Message Service	57
<i>Klaus Jank, Roy Oberhauser</i>	
Web Services-Based Transactional Workflows – Advanced Transaction Concepts	70
<i>Kai Güntzel</i>	
MIDAS's Routing and Scheduling Approach for the Australian Transport Industries	83
<i>Manish Malhotra, Zahir Tari</i>	
An Approach to the Process Maps Based on Semantic Web Methodologies	98
<i>Roberto Fresco, Andrea Pederiva</i>	
PEPt – A Minimal RPC Architecture	109
<i>Harold Carr</i>	

Modelling Organic Adaptable Service-Oriented Enterprise Architectures	123
<i>George R. Ribeiro-Justo, Tereska Karran</i>	
Towards a Semantic Integration of Medical Relational Databases by Using Ontologies: A Case Study	137
<i>Tom Deray, Pieter Verheyden</i>	
Workshop on Human Computer Interface for Semantic Web and Web Applications (HCI-SWWA)	
Workshop PC Chairs' Message	151
<i>Tharam S. Dillon, Elizabeth Chang</i>	
Designing Ontology-Based Interactive Information Retrieval Interfaces	152
<i>Elena García, Miguel-Ángel Sicilia</i>	
Usability Issues Confronting Mobile Devices as Internet Interfaces for General Purpose Navigation	166
<i>José María Gutiérrez, Roberto Barchino, José Antonio Gutiérrez de Mesa</i>	
Navigating through the RLATES Interface: A Web-Based Adaptive and Intelligent Educational System	175
<i>Ana Iglesias, Paloma Martínez, Fernando Fernández</i>	
Searching with Semantics: An Interactive Visualization Technique for Exploring an Annotated Image Collection	185
<i>Paul Janecek, Pearl Pu</i>	
Ontology-Mediated Service Matching and Adaptation	197
<i>Han-Kuan Yu, Tse-Ming Tsai, Yong-Fang Yang, Wen-Shi Huang</i>	
The Role of Vague Categories in Semantic and Adaptive Web Interfaces	210
<i>Miguel-Ángel Sicilia</i>	
Two Layer Web User Interface Analysis Framework Using SNN and iFIN	223
<i>William Gardner, Elizabeth Chang, Tharam S. Dillon</i>	
Usability Metrics for E-learning	235
<i>Siu Keung Wong, Thao Thi Nguyen, Elizabeth Chang, Nimal Jayaratna</i>	

Creating a “DOGMatic” Multilingual Ontology Infrastructure to Support a Semantic Portal	253
<i>Jan De Bo, Peter Spyns, Robert Meersman</i>	
Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints	267
<i>Dirk Draheim, Gerald Weber</i>	
Semantic Web Services for Smart Devices in a “Global Understanding Environment”	279
<i>Vagan Terziyan</i>	
Construction of Generic Web-Based User Interfaces	292
<i>Arno Puder</i>	
A Word Sense Disambiguation Algorithm for Information Retrieval Applications	306
<i>Giuliano Pascucci, Sandro Spadaro</i>	
Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES)	
Workshop PC Chairs’ Message	318
<i>Angelo Corsaro, Ron K. Cytron, Corrado Santoro</i>	
Towards a High Integrity Real-Time Java Virtual Machine	319
<i>Hao Cai, Andy J. Wellings</i>	
Aspect-Orientation of Control Application Code for Distributed Automation Systems: The TORERO Approach	335
<i>Marcus Tangemann, Christian Schwab, Athanasios P. Kalogeras, Kai Lorentz, Aggeliki S. Prayati</i>	
JOP: A Java Optimized Processor	346
<i>Martin Schoeberl</i>	
Processing Group Parameters in the Real-Time Specification for Java	360
<i>A. Burns, Andy J. Wellings</i>	
Doing Firm-Real-Time with J2SE APIs	371
<i>Kelvin Nilsen</i>	
Container Model Based on RTSJ Services	385
<i>Ruth Tolosa, José P. Mayo, Miguel A. de Miguel, M. Teresa Higuera-Toledano, Alejandro Alonso</i>	
Reference Objects for RTSJ Memory Areas	397
<i>Andrew Borg, Andy J. Wellings</i>	

Deriving Java Virtual Machine Timing Models for Portable Worst-Case Execution Time Analysis	411
<i>Erik Yu-Shing Hu, Andy J. Wellings, Guillem Bernat</i>	
Efficient Bytecode Verification Using Immediate Postdominators in Control Flow Graphs	425
<i>Cinzia Bernardeschi, Nicoletta De Francesco, Luca Martini</i>	
Computational Reflection for Embedded Java Systems	437
<i>Antonella Di Stefano, Marco Fargetta, Emiliano Tramontana</i>	
Issues in the Design and Implementation of Efficient Interfaces between Hard and Soft Real-Time Java Components	451
<i>Kelvin Nilsen, Andrew Klein</i>	
The Metronome: A Simpler Approach to Garbage Collection in Real-Time Systems	466
<i>David F. Bacon, Perry Cheng, V.T. Rajan</i>	
Memory Management Design to the Concurrent Execution of RTSJ Applications	479
<i>M. Teresa Higuera-Toledano</i>	
Experiences in Adopting Real-Time Java for Flight-Like Software	490
<i>Edward G. Benowitz, Albert F. Niessner</i>	
A Patterns Catalog for RTSJ Software Designs	497
<i>Edward G. Benowitz, Albert F. Niessner</i>	
RTSJ Memory Areas and Their Affects on the Performance of a Flight-Like Attitude Control System	508
<i>Albert F. Niessner, Edward G. Benowitz</i>	
Scheduling Policy Costs on a JAVA Microcontroller	520
<i>Leomar S. Rosa Jr., Flávio R. Wagner, Luigi Carro, Alexandre S. Carissimi, André I. Reis</i>	
Experiences and Benchmarking with JTime	534
<i>Greg Bollella, Krystal Loh, Graham McKendry, Thomas Wozenilek</i>	
Utility Accrual Scheduling with Real-Time Java	550
<i>Shahrooz Feizabadi, William Beebe Jr., Binoy Ravindran, Peng Li, Martin Rinard</i>	
A C++ Native Interface for Interpreted JVMs	564
<i>Angelo Corsaro, Corrado Santoro</i>	

Workshop on Regulatory Ontologies and the Modelling of Complaint Regulations (WORM CoRe)

Workshop PC Chairs' Message.....	578
<i>Mustafa Jarrar, Anne Salaun</i>	
On the Ontology of Digital Identification.....	579
<i>Giles Hogben, Marc Wilikens, Ioannis Vakalis</i>	
Ontology-Based Customer Complaint Management	594
<i>Mustafa Jarrar, Ruben Verlinden, Robert Meersman</i>	
Some Ontological Tools to Support Legal Regulatory Compliance, with a Case Study	607
<i>Aldo Gangemi, Alessandra Prisco, Maria-Teresa Sagri, Geri Steve, Daniela Tiscornia</i>	
Regulatory Ontologies: An Intellectual Property Rights Approach	621
<i>Jaime Delgado, Isabel Gallego, Silvia Llorente, Roberto García</i>	
Copyright Conflicts Resolution: A Method Based on Legal Ontologies in an Architecture Enabling Subsidiarity	635
<i>Melanie Dulong de Rosnay</i>	
A Comparison of a Regulatory Ontology with Existing Legal Ontology Frameworks	648
<i>John Kingston, Wim Vandenberghe</i>	
Representing Multilingual and Culture-Specific Knowledge in a VAT Regulatory Ontology: Support from the Termontography Method	662
<i>Koen Kerremans, Rita Temmerman, Jose Tummers</i>	
Ontology-Based Platform for Trusted Regulatory Compliance Services.....	675
<i>Henry Ryan, Peter Spyns, Pieter De Leenheer, Richard Leary</i>	
Business Contract Obligation Monitoring through Use of Multi Tier Contract Ontology	690
<i>Vandana Kabilan, Paul Johannesson, Dickson M. Rugaimukamu</i>	

Workshop on Metadata for Security (WMS)

Workshop PC Chairs' Message.....	703
<i>Ernesto Damiani</i>	
Efficient Processing of Secured XML Metadata	704
<i>Ling Feng, Willem Jonker</i>	
Multimedia Access Control Using RDF Metadata	718
<i>Naren Kodali, Csilla Farkas, Duminda Wijesekera</i>	

Managing Identities via Interactions between Ontologies	732
<i>Paolo Ceravolo</i>	
Implementing an Access Control System for SVG Documents	741
<i>E. Fernández-Medina, G. Ruiz, S. De Capitani di Vimercati</i>	
Administration Model for Or-BAC	754
<i>Frédéric Cuppens, Alexandre Miège</i>	
Metadata Representation for Semantic-Based Multimedia Security and Management	769
<i>Yuriy Rytsar, Sviatoslav Voloshynovskiy, Thierry Pun</i>	
Comparing Logic-Based and XML-Based Rights Expression Languages	779
<i>Cheun Ngen Chong, Sandro Etalle, Pieter H. Hartel</i>	
Simplifying CORBA Security Service to Support Service Level Access Control	793
<i>Atul Kumar, Pankaj Jalote, Deepak Gupta</i>	
The Role of Deontic Concepts and Commitment in the Policy Specification of Mobile Computing Environment	807
<i>Xiaofeng Gong, Julian Newman</i>	
Business to Consumer Markets on the Semantic Web	816
<i>Robert Tolksdorf, Christian Bizer, Rainer Eckstein, Ralf Heese</i>	
Modeling Business Collaborations in Context	829
<i>Birgit Hofreiter, Christian Huemer</i>	
Security Framework for Mobile Agent Platforms (SFMAP)	845
<i>Terry Kapnoullas, Elizabeth Chang, Tharam S. Dillon, Ernesto Damiani</i>	
An Architecture for Software Agreements between Autonomous Agents	859
<i>Giles Hogben</i>	
Workshop on Reliable and Secure Middleware (WRSM)	
Workshop PC Chairs' Message	870
<i>Priya Narasimhan, Brian Blake, Giacomo Cabri</i>	
Fault-Tolerant Middleware	
Fairly Redistributing Failed Server Load in a Distributed System	871
<i>Venkatesh Sangam, Christopher B. Mayer, K. Selçuk Candan</i>	

Aquarius: A Data-Centric Approach to CORBA Fault-Tolerance 885
*Gregory Chockler, Dahlia Malkhi, Barak Merimovich,
David Rabinowitz*

FT-JAVA: A Java-Based Framework for Fault-Tolerant
Distributed Software 899
Vicraj Thomas, Andrew McMullen, Lee Graba

Quality of Service

Adaptive Message Packing for Group Communication Systems 912
*Alberto Bartoli, Cosimo Calabrese, Milan Prica,
Etienne Antoniutti Di Muro, Alberto Montresor*

DADO: A Novel Programming Model for Distributed, Heterogenous,
Late-Bound QoS Implementations 926
Eric A. Wohlstadter, Premkumar T. Devanbu

Many Aspects of Reliabilities in a Distributed Mobile Messaging
Middleware over JMS 934
Eiko Yoneki

Enterprise Middleware Issues

A Flexible Middleware Platform with Piped Workflow 950
Wanjun Huang, Uwe Roth, Christoph Meinel

Using Web Services Technology for Inter-enterprise Integration of
Digital Time Stamping 960
A. Cilardo, A. Mazzeo, L. Romano, G.P. Saggese, G. Cattaneo

A Lightweight Security Model for WBEM 975
*Giuseppe Cattaneo, Luigi Catuogno, Umberto Ferraro Petrillo,
Ivan Visconti*

Agent-Based Middleware

Security and Trust in Agent-Oriented Middleware 989
Agostino Poggi, Michele Tomaiuolo, Giosuè Vitaglione

A Conceptual Framework for Web Services Semantic Discovery 1004
Islam Elgedawy

A Holonic Multi-agent System for Robust, Flexible, and Reliable
Medical Diagnosis 1017
Rainer Unland

Ph.D. Presentations

Choosing Reputable Resources in Atomistic Peer-to-Peer Environments	1031
<i>Georgios Pitsilis</i>	
A Fast Group Communication Mechanism for Large Scale Distributed Objects	1036
<i>Hojjat Jafarpour, Nasser Yazdani</i>	
Application of Catalogues to Integrate Heterogeneous Data Banks.....	1045
<i>Zsolt T. Kardkovács, Gábor M. Surányi, Sándor Gajdos</i>	
Certificate Validation in Untrusted Domains.....	1057
<i>Omar Batarfi</i>	
Author Index	1069

On Enhancing Component-Based Middleware with Transactions

Marek Prochazka¹, Romain Rouvoy², and Thierry Coupaye³

¹ INRIA Rhône-Alpes, 665 avenue de l'Europe, 38334 Saint Ismier Cedex, France

² INRIA Jacquard Project, LIFL, 59655 Villeneuve d'Ascq Cedex, France

³ France Télécom R&D, 28 Chemin du Vieux Chêne, BP 98, 38243 Meylan Cedex, France

Abstract. It is believed that transactions belong to key services of component-based middleware. However, surprisingly, there is no general agreement on how the component-based middleware support for transactions should look like. In this paper we exploit our experiences with Jironde [4], a flexible framework that extends the Fractal component model [1, 2] with transactions via a set of transactional controllers that manage transactions on behalf of a component. Then we identify several key architectural and technical issues related to enhancing component-based middleware with transactions. We investigate how current technologies address these issues, and the suitability of current standards to support transaction processing in component-based middleware.

1 Transactions and Components: Architectural Issues

Different component models deal with component's participation in a transaction differently. In the *explicit transaction participation*, the scenario of involving a component C to a transaction t essentially consists of the three steps as follows: 1) C is registered to t . 2) A client invokes various operations on C . 3) At the time of t 's validation, the transaction manager invokes specific methods of the registered C 's interfaces. These (callback) methods must be implemented by the transactional components. With the *implicit transaction participation*, components are not obliged to implement any functionality related to transactions. Any time C is visited by a transaction t , the transaction manager of the container keeps all necessary information to manage atomicity, concurrency control, and recovery. Different component standards deal with component participation in transactions differently. CCM use the explicit transaction participation, COM+ uses the implicit one, while EJB mix both.

Component-unaware transactions manipulate data without any knowledge on whether they are organized or related to components. For *component-aware transactions*, components are the data they manipulate with. We believe that all the CCM, EJB, and COM+ transactions are component-unaware. A component is *transaction-unaware* if its code does not use any transactional primitives and is not therefore in any way dependent on any transactional standard, while a *transaction-aware* component is the opposite. Component's awareness of transactions reflects the implicit/explicit transaction participation but includes also some hidden expectations of the component design. *Transaction-unaware container* does not deal with

transactions, which are managed at the application level instead. *Transaction-aware container* provides some transaction management features, such as container-demarcation, transaction propagation, concurrency control, etc. EJB, CCM and COM+ are examples that provide such containers.

2 Transactions and Components: Technical Issues

The technical issues related to transactional component include concurrency control, recovery, and transaction context propagation. As for concurrency control, the current technologies either use a simple read/write model or do not allow any concurrent access to component instances at all. It would be beneficial to exploit the concurrency potential of components by e.g. the use of conflict matrixes defined on all the methods of all implemented interfaces. Both concurrency control and recovery of components reflect the architectural patterns presented in the previous section. Another important issue is the transaction context propagation. There are several options of how to specify a *transaction propagation policy*, as well as whether to define it either during the component's development or during its deployment. A last issue is to support the definition of new propagation policies, like JOTDF [5] does.

3 Conclusion

Our experiments with Fractal and Jironde have shown that the coexistence of components and transactions raises more non-trivial architectural and technical issues than expected. The current middleware standards and technologies do not address these issues satisfactorily. So we hope that future developments will take into account such issues for transactions to remain a key service of component-based middleware. Details on our work are available at <http://jotm.objectweb.org>.

References

1. Bruneton, E., Coupaye, T., Stefani, J.-B., "Recursive and Dynamic Software Composition with Sharing", the 7th International Workshop on Component-Oriented Programming (WCOP 2002, in conjunction with ECOOP), Malaga, Spain (2002)
2. ObjectWeb, "The Fractal Composition Framework Specification", Version 1.0, <http://fractal.objectweb.org/> (2002)
3. Prochazka, M., "Advanced Transactions in Component-Based Software Architectures", Ph.D. thesis, Charles University, University of Evry (2002)
4. Prochazka, M.: "Jironde: A Flexible Framework for to Make Components Transactional", 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2003), Paris, France, to appear (2003)
5. Rouvoy, R., Merle, P., "Abstraction of Transaction Demarcation in Component-Oriented Platforms", ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil (2003)

Online Consistency Checking for Replicated Objects^{*}

Giovanni Masarin¹, Alberto Bartoli¹, and Vance Maverick²

¹ Dip. Elettrotecnica, Elettronica ed Informatica, Università di Trieste (Italy).

² Dipartimento di Scienze dell'Informazione, Università di Bologna (Italy).

Abstract. Enabling an application to automatically test its correctness even when deployed in a production environment can be a very effective way to improve its dependability. In this paper we are concerned with the live verification of the consistency of a replicated system, a topic that has received little attention from the research community so far. In particular, we consider how to check efficiently whether the replicas of a potentially large set of objects are indeed being kept in sync. We propose several solutions for implementing this check, and analytical models for predicting their performance. We enumerate a set of parameters characterizing the operating environment and the specific application, and analyze the effect of these parameters on performance, under each solution. With realistic parameter values, this analysis provides significant insights into performance. For example, it can help decide whether on-line verification can be used under demanding production conditions, or only in development and testing.

1 Introduction

Recent analysis of failure data at large Internet sites [4] has suggested that functionalities of *on-line testing* could be the most effective approach in preventing service outages. Equipping the system with the ability to automatically test its correctness in the production environment makes it possible, for example, to detect failures that can hardly be detected before production — e.g., scale-dependent or configuration-dependent failures. We feel that very similar conclusions can be drawn for *object replication* systems (e.g., [3,5]), because replication certainly adds complexity to the system. Ideally, the system should be able to detect automatically whether the invariants defining the correctness of object replication are violated and in that case to take the suitable corrective actions — e.g., if one of the replicas is out-of-sync, bringing it off-line and reboot it. Such a feature could improve reliability of the overall application, which is especially important when the object replication system is part of a complex and large agglomeration of components meant to run 24x7. In particular, because replication is used precisely for enhancing reliability and availability.

Reasons why the replication system might not work as expected include errors in the replication algorithm or its implementation that remain latent most of the time as they depend on very rare system states. Other reasons include possible mismatch between hypothesis and operating environment [1]. For example, the replication algorithm could require that all replicas of an object behave deterministically whereas the interactions

^{*} This work is supported by Microsoft Research (Cambridge, UK) and the EU IST Programme (Project ADAPT IST-2001-37126).

with the operating system could return different results at different nodes, perhaps because the system is running 24x7 and is subject to complex and unknown software aging effects [2].

The live verification of the consistency of a replicated system, however, has received very little attention from the research community so far. In this paper we have considered the problem of how to check efficiently whether the replicas of a potentially large set of objects are indeed being kept in sync or not. We propose several solutions for implementing this check, that differ in the amount of parallelism exploited in the comparison and in the synchronization steps required between replicas. For each solution, we have identified the factors that may affect the duration of the check and developed an analytical model for predicting such duration. Some factors depend on the operating environment (e.g., the network bandwidth) while some others depend on the specific application (e.g., how long it takes to compare two objects for equality). We have performed many experiments for determining suitable values for the application-specific parameters with respect to the replication of session objects in a Java-based web server (Tomcat). We have coded the analytical models in a spreadsheet. This simple tool has enabled us to gain significant insights into both the absolute and relative cost of the various solutions, to understand which parameters are going to have more substantial impact on the overall performance and to devise in which environments a given solution could be preferred to the others. Full details can be found in the companion report (available from: <http://webdee.univ.trieste.it/Archivio/Docenti/Bartoli/DOA-03-full.pdf>). An important finding is that the simplest Centralized solution, where one designated node fetches all the necessary data and then performs the entire comparison, may indeed work well. An implementation of the Centralized solution in a 3-way replicated web service shows that the prediction of the model indeed matches the actual performance (e.g., approximately 180 ms for a system with 200 clients). The prototype performs the check repeatedly every T seconds. Surprisingly, neither throughput nor latency are affected for $T \geq 5sec$. With more frequent checks ($T \in [1, 5]sec$), performance never drops below 5%. These preliminary results indicate that the proposed approach is indeed practically viable.

References

1. K. Birman. A review of experiences with reliable multicast. *Software: Practice & Experience*, 29(9):741–774, July 1999.
2. V. Castelli, R. E. Harper, P. Heidelberger, S. Hunter, K. Trivedi, K. Vaidyanathan, and W. P. Zeggert. Proactive management of software aging. *IBM Journal of Research & Development*, 45(2), March 2001.
3. P. M. Melliar-Smith, L. E. Moser, and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.
4. D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do internet services fail, and what can be done about it? In *4-th USENIX Symposium on Internet Technologies and Systems*, March 2003.
5. Y. Ren, D. Bakken, T. Courtney, M. Cukier, D. Karr, P. Ruble, C. Sabnis, W. Sanders, R. Schantz, and M. Seri. AQUA: An adaptive architecture that provides dependable distributed objects. *IEEE Transactions on Computers*, 52(1):31–49, January 2003.

Integrating Container Services with Pluggable System Extensions

Andreas Leicher¹, Alexander Bilke¹, Felix Bübl², and E. Ulrich Kriegel³

¹ Technische Universität Berlin, Germany
`{aleicher|bilke}@cs.tu-berlin.de`

² Imphar AG, Berlin, Germany
`felix.buebl@imphar.com`

³ Fraunhofer ISST, Berlin, Germany
`Ulrich.Kriegel@isst.fhg.de`

A common problem in today's software is the lack of separation of concerns, which results in intertwined code, that is difficult to maintain. In order to make a component-based system adaptive to new or changed requirements, distinct concerns must be separated during both design and implementation. Common component platforms support separation for several concerns by providing a number of standard container services, e.g. persistence and transaction management. However, complete separation of concerns require the integration of purpose-build container services.

Today, requirements of a system are frequently changing whereby one requirement can affect several components. In order to enforce requirements, custom-made services can be inserted into the communication paths of the involved components. Dynamic configuration management is required to allow flexible configuration of rules that govern the communication of components within a system.

The integration of new services can hardly be achieved in an Enterprise JavaBeans (EJB) environment, whose specification does not allow for extending the component container. The *EJBComplex* framework has been developed to overcome this restriction. It has been implemented on application level and is, thus, vendor independent. The *EJBComplex* framework consists of *components* and *pluggable extensions*. One such extension, called *plug-in*, contains the implementation of a single service. To facilitate implicit invocation of the services, method calls must be intercepted before reaching the component. As in several other systems, a *proxy*, which encapsulates the component, is used for this purpose. The framework described here has the following key features:

Dynamic Configuration. The configuration of EJB container services has to be done once prior to deployment. Changes to the system requirements with respect to these services call for a re-configuration of the system. On the contrary, the *EJBComplex* framework facilitates the dynamic configuration of proxies. New plug-ins can seamlessly be integrated into the running system, and the components' proxies can be reconfigured without affecting their availability. This can be done using the *proxy manager*, which is part of the framework and is designed for the management of proxies and plug-ins.

Complex execution order. Many approaches to separation of concerns, e.g. AspectJ or composition filters, assemble the implementation of the concerns sequentially. Unfortunately, in many cases the concerns are not orthogonal, i.e. there can be a functional overlap between two or more concerns. This can lead to side effects, which are commonly known as *feature interactions*. This problem is regarded in the design of the framework by allowing a more complex combination of plug-ins. Plug-ins can be arranged hierarchically in a tree structure. In general, those plug-ins implementing a concern can be found at the leafs of the tree and are, with the exception of simple conditions (see below), independent of the system in which they are integrated. The inner nodes of the tree contain *controller plug-ins* which govern the execution of those plug-ins on the next hierarchy level that are connected to the controller. The implementation of a controller contains knowledge of interactions that might occur in a given system and is, thus, system dependent.

Conditional execution. Requirements can demand conditional execution of services. In order to support configuration management, we add Event Condition Action rules on top of plug-ins. Thus, services become independent of changes in the requirements. A plug-in evaluates system dependent properties and decides if a service has to be applied.

Configuration Management via Context-Based Constraints. Many plug-ins at different communication paths can be needed in order to enforce one requirement. In order to determine which plug-in(s) must be deployed at which communication path(s), the components are annotated with formatted metadata called ‘context properties’. A *context-based constraint* (CoCon) can indirectly select its constrained elements according to their context property values. It relates *two sets* of components and expresses a condition for each *pair* of related components. One CoCon can constrain several communication paths. The constrained communication paths can automatically be identified by evaluating the CoCon’s context condition. Plug-Ins must be deployed at each communication path between each pair of constrained components.

The indirect selection of constrained components is particularly helpful in highly dynamic or complex systems. Every new, changed or removed component is automatically constrained by a CoCon according to the component’s context property values. A flexible framework is needed in order to enforce CoCons at runtime, because both contexts and requirements can change at runtime. Such a framework needs all the features provided by the EJBCComplex described in [1]: it needs complex execution order of plug-ins, conditional execution of plug-ins, as well as dynamic configuration of plug-ins.

References

1. Andreas Leicher, Alexander Bilke, Felix Bübl, and Ulrich Kriegel. Integrating Container Services with Pluggable System Extensions. Technical report, Technische Universität Berlin, Germany, 2003. to appear.

SGI-SAT Trillo: A Full Scope Simulator for Nuclear Power Plants

Manuel Díaz and Daniel Garrido

Department of Languages and Computing Science
University of Málaga, Spain
{mdr, dgarrido}@lcc.uma.es

Abstract. The application of new technologies and programming tools represents a challenge and an economic risk for companies, which not all are prepared to assume. This paper presents the application of new technologies and methodologies like CORBA and UML in the development of software for a simulator of the Control Room of a Nuclear Power Plant used for the training of future operators in a safe way. The developed software has allowed the adaptation of previous simulation software to these new methodologies and standards; and the creation of new applications, aiming at the building of reusable components in future projects.

1 Introduction

The continuous appearances of new technologies, methodologies, tools and programming languages like CORBA [1], UML [2], .NET [3] or EJB [4], makes companies carefully consider the adoption of some of these new elements in their projects.

The present paper is situated within the project of simulators for the Trillo Nuclear Power Plant that during two years has allowed the carrying out of a simulator in full scope of the Control Room of the Trillo Nuclear Power Plant sited in Guadalajara (Spain). The simulator is an exact replica of the Control Room of the Power Plant, taking care of all details, from physical artifacts like furniture, control panels, etc. to software, simulating the applications running in the Power Plant. The core of the simulation are the simulation models which proportionate the values of the distinct signals and variables providing the sensation of being in the real Control Room. The principal intention of this simulator is the training of future operators, allowing for the practice of different situations, from the most usual like temperature monitoring, valve manipulation, etc. to most unusual situations like emergency situations that logically cannot be practiced in the real Control Room.

The software described in this paper has been carried out in a shared project between the company Tecnatom S.A. and the department of Languages and Computing Science of the University of Málaga. The work has been principally related to the adaptation to CORBA and UML of previous software and in the creation of new ap-

plications needed for this concrete simulator, with special emphasis on the building of software components [5] that could be reused in future projects.

The rest of the paper is organized as follows: The hardware and software architectures of the simulator are presented in section 2. Section 3 details implementation aspects of the developed software, like code reusing, CORBA and software componentization. The paper finishes with some conclusions and new works.

2 System Overview

The Trillo Simulator Project is really divided into two profoundly related simulators that determinate the hardware architecture and the physical infrastructures. The first simulator is the denominated Interactive Graphic Simulator (SGI), which through graphic applications allows the training of future operators. The second simulator is the denominated Total Scope Simulator (SAT) being this the full scope simulator and exact replica of the Control Room of the Power Plant. The software of both simulators is very similar, and the code reusing plays a fundamental role [6].

The hardware components of SAT and SGI are principally formed by a Simulation Computer (responsible for the carrying out of the simulation process), Instructor Console (used by the instructor, it allows the creation of simulation conditions), Physical panels (exact replicas - 17x18 m.) and some others simulated subsystems. The hardware equipment of the SGI simulator additionally includes the needed for the student posts of the simulator. SGI and SAT simulators must work with about 26,000 signals of analogue and digital types.

The developed software for the SGI-SAT project is divided into three well differentiated phases. First of all, the adaptation and documentation of previous software was performed for its use with CORBA, UML and object-oriented methodologies. Secondly, the development of the new applications needed for this concrete project, like the applications related to the simulation of the Plan Process Computer (*PPC*) or the *SGI Displays*. The third phase consisted of the interconnection of the updated software with the new applications.

There are two parts in the final software architecture communicated through CORBA. The server part includes the following main components: Simulation engine (execution of simulations models and variables updating), Simulation Database, *Simcorba* (“Simulation Server” for clients, it offers a set of services like periodic sending of variables, updating of variables, etc.).

The client part includes the rest of components: *Receiver* (responsible for the communication between *Simcorba* and the rest of applications) and different applications like *DESI* (Variables debugger), *SPV* (online modification of simulation models), *SGI Displayer* (simulation of physical panels, it allows their manipulation) or *PPC* (monitoring of data, alarms and reports).

3 Implementation Issues

There are three important aspects in the code implementation: reusing previous code, CORBA utilization and the aiming to the software componentization.

Two different development environments were used: Unix and Windows. In Unix the Operating System IRIX 6.5 of Silicon Graphics, with the C++ compiler MIPSpro 7.2.1 were used. On the other hand, Windows NT and Windows 2000 with Microsoft Visual C++ 6.0 were used on Windows.

In the case of CORBA the implementation initially chose was Orbix 3.0.1 of Iona for C++. Later a porting to TAO 1.2. [7] was performed. TAO is a freely available CORBA ORB that due to characteristics like predictable timing and robustness is very adequate for its using in real-time applications.

All the software development process is aimed by practice of software engineering through UML with tools like Rational Rose [8]. And finally, for the development of the graphical sheets of the *SGI Displays*, the tool DataViews 2.1 [9] was chosen.

In the case of the simulator core, there was previous software developed on Unix and principally coded in C, FORTRAN and even with assembly code. Through the porting process, the simulators were updated to the object-oriented paradigm with all the advantages that it offers. Furthermore, to facilitate the interoperability with tools and applications, a communications layer with CORBA was created.

In the case of new developed components and actually reused, the *Receiver* component and *Simcorba* are very important. The *Receiver* component can be used in a wide range of applications. Its principal purpose is to actuate like a passive data container, receiving data from simulation (*Simcorba*). There are two versions for the pair *Simcorba/Receiver*. The first version used the distributed callbacks model of CORBA [1]. The *Receiver* components were developed thinking in their reusing and in the hiding of the code related to communications. The second version for the couple *Simcorba/Receiver* was carried out on TAO using the Real-Time Event Service offered by TAO [10]. In this case, there was a higher flexibility allowing for a higher uncoupling between the data sending process (*Simcorba*) and the data receivers (*Receiver*). The *Event Service version* is more easy to maintain and more flexible than the *Callbacks version* where other aspects like lost connections, client references, etc. make the implementation more difficult. On the other hand, the *Event Service version* entirely depends on the implementation of the Event Service.

A principal goal in the SGI-SAT project is the creation of software components that can be reused in future projects. All the applications and libraries were developed with this vision in mind, obtaining a higher level of software componentization, specially with the flexibility and reusing obtained with the communications libraries. Any standard component model like COM [12] or Java Beans [4] has been used, except CORBA when it has been possible. The use of CCM [13] had been desirable, but during the development of the project, the existing implementations didn't guarantee the success of the project.

In general, the higher componentization levels have been obtained in the communication areas thanks to CORBA, that even without considering CCM it is a valuable help in the creation of software components.

4 Conclusions

The adoption of new technologies in a market dominated by costs requirements, development time, etc. is a big problem for companies. In this paper, the adoption of new technologies and methodologies like CORBA and UML has been presented in a project for the carrying out of a Nuclear Power Plant Simulator. It has been verified that the use of CORBA and UML has provided a good experience, and its utilization in large software projects is feasible.

The developed software is being reused in new projects like *Javi* (graphical Displayer) implemented on Java-3D, using a great number of the components developed in the present project.

The simulator is now in its final testing phases, and the beginning of its real using is scheduled at the ending of 2003.

Acknowledgements. We would like to acknowledge the help from the people of the División de Simulación at Tecnatom S.A. who contributed to the creation of this paper.

References

1. Henning, M., Vinoski, S.: Advanced CORBA Programming with C++. Addison-Wesley Longman (1999)
2. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language User Guide. Addison-Wesley Professional (1998)
3. Holm, C. et al: Professional .NET Framework. Wrox Press (2001)
4. Monson-Haefel, R.: Enterprise JavaBeans. 3rd edn. O'Reilly & Associates (2001)
5. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman (1999)
6. Sametinger, J.: Software Engineering With Reusable Components. Springer Verlag; (1997)
7. Levine, D.L., Mungee S., Schmidt, D.C.: The Design of the TAO Real-Time Object Request Broker. Computer Communications 21, pp. 294–324, 1998. 8
8. Booch, G., Quatrani, T.: Visual Modeling with Rational Rose 2000 and UML. 2nd Edition. Addison-Wesley Pub Co (1999)
9. GE Fanuc Automation North America, Inc.: <http://www.gefanuc.com/dataviews/>
10. Harrison, T.H., Levine, D.L., Schmidt, D.C.: The Design and Performance of a Real-Time CORBA Event Service. Proceedings of the OOPSLA '97 conference, Atlanta, Georgia.
11. Armstrong, T.: Designing and Using ActiveX Controls. Hungry Minds (1996)
12. Box, D.: Essential COM. Addison-Wesley, Reading, MA (1997)
13. O'Ryan C., Schmidt, D.C., Wang, N.: Overview of the CORBA Component Model in Component-Based Software Engineering, Addison-Wesley (2000).

MobJeX: A Declaratively Configurable Java Based Framework for Resource Aware Object Mobility

Caspar Ryan and Stephen Perry

RMIT University, School of CS & IT
PO Box 71, Bundoora
Melbourne, Australia, 3083
{caspar, sperry}@cs.rmit.edu.au

Abstract. MobJeX (Mobile Java Objects) is a Java based resource aware adaptive code mobility framework that is currently under development. MobJeX differs from its predecessors such as Sumatra [1] and FarGo [2, 3] by providing support for the transparent distribution of application code to client devices in a heterogenous Next Generation Internet (NGI) environment [4]. This paper briefly discusses the architecture of MobJeX.

1 Declarative Rule Based Application Configuration

MobJeX uses an XML based application deployment descriptor to specify rule-based configuration for object grouping and migration policies. This strategy was chosen to provide greater transparency, and less development effort, than a script-based approach. Although the declarative approach cannot ultimately match the customisability of scripting, it can provide a significantly greater amount of system support and automation. As such, the developer can write applications with a greater level of transparency, with the deployer tuning the system by specifying application specific policy rules for object grouping and migration.

2 Dynamic Object Mobility

MobJeX treats object mobility as a dynamic property, which can vary depending upon the state of the computational environment, as reported by the resource monitor (see section 4). Object mobility rules in the deployment descriptor are specified hierarchically, ranging from abstract application level rules to specialised object level rules. As such, a deployer need only specify application level rules in order to facilitate dynamic object mobility and client/server adaptation in a MobJeX environment. Optimising the application for a specific client is accomplished by specifying per object deployment rules and specific object grouping strategies. Additionally, further optimisation can be achieved by explicitly specifying object groups (by naming the objects comprising a group) and co-location constraints (via declarative resource requirement rules). The use of object mobility rules echoes the hard-coded relocation semantics of FarGo's *complet* approach with less developer effort and without the need to recompile code when adaptation policies change.

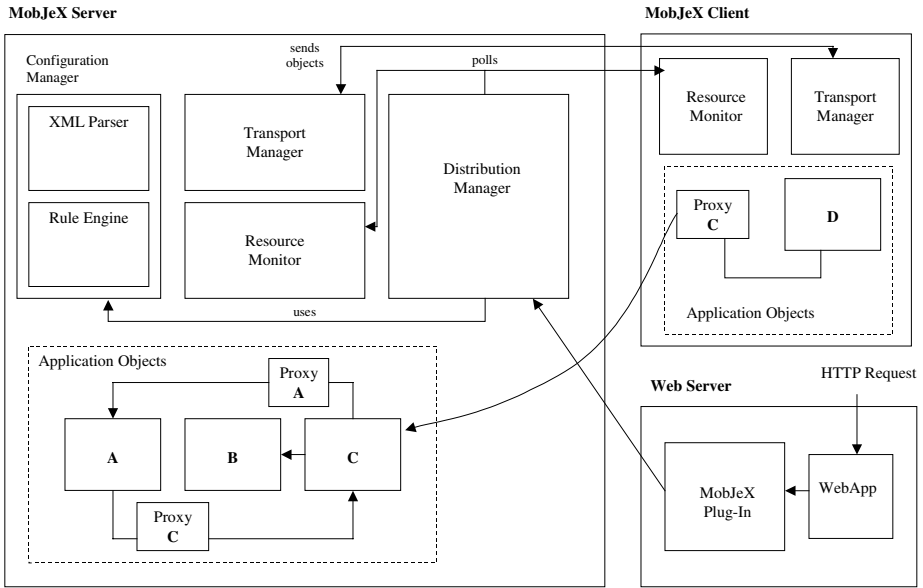


Fig. 1. Overview of MobJeX Architecture

3 Pluggable Resource Monitoring

Like Sumatra, MobJeX provides a resource monitoring API using familiar JavaBean event handling semantics. In order for a device to participate as a client in the MobJeX environment, it must have a resource monitoring implementation that can, at the minimum, report available memory. Although such limited reporting would affect performance, it does enable individual objects or object groups of specified memory requirements to be migrated and executed. Nevertheless, client specific implementations of the MobJeX resource monitoring API should, at a minimum, aim to report basic memory, networking, and CPU information. The authors have developed an initial resource monitoring implementation based on a Java Native Interface (JNI) to the Microsoft Windows performance API.

4 MobJeX Architecture

MobJeX uses a proxy-based object tracking strategy, which is comparable to FarGo, but operates in a simpler client/server fashion. This implementation decision was made primarily to simplify object communication, and more readily facilitate dynamic object grouping and mobility, but has the incidental benefit of being a more secure and trusted model for interaction in an NGI environment. Furthermore, our scheme uses a simpler client-side transport manager that is more suitable for use on limited devices. Fig. 1 depicts the high level architecture of MobJeX, the main components of which

are the server based *configuration manager* and *distribution manager*, and client and server *resource monitors* and *transport managers*. Note that distribution decisions, based on information received from the configuration manager, are made on the server. Nevertheless, clients still require a lightweight transport manager, which facilitates the physical mobility and proxy resolution of migrated objects, as well as a client specific resource monitor.

Application objects that meet the following criteria are accessed via an object proxy that either refers directly to the local object, or in the case of remote objects, knows their location, and provides the means to access them remotely. The circumstances in which proxies are used are: 1) If an object can move, all objects it refers to are proxied, excepting the case where explicit co-locality constraints are present. 2) If a non-moveable object refers to objects that can move, it must also access those objects via a proxy, so that they can be accessed upon migration. Note that the moveability of an object is determined by evaluating the mobility rules of the deployment descriptor via the configuration manager. Furthermore, it is possible to declare at the application level that objects either are, or are not, moveable by default.

5 Example

A typical example of a MobJeX enabled client could involve an initial HTTP request asking for a Java Swing based user interface object. Once delivered to the client and executed, this object would interact with other application objects, either locally or remotely, according to the specified application requirements and adaptation policies. A non-MobJeX enabled client would also initiate its interaction via the web front end, but unlike its MobJeX counterpart, would continue to do so for the lifetime of the session. In this case, all processing would occur on the server, with responses taking the form of a standard mark-up based user interface delivered via standard HTTP responses. In terms of underlying protocols, MobJeX is currently using Java RMI as the underlying transport mechanism, necessitating firewall configuration and the use of Java specific security mechanisms. Currently under consideration is whether an alternative such as SOAP-RPC can provide a more flexible and secure approach.

References

1. Acharya, A., M. Ranganathan, and J. Saltz, *Sumatra: A Language for Resource-aware Mobile Programs*, in *Mobile Object Systems: Towards the Programmable Internet*, C. Tschudin, Editor. 1997, Springer-Verlag: Heidelberg, Germany. p. 111–130.
2. Holder, O., I. Ben-Shaul, and H. Gazit, *System Support for Dynamic Layout of Distributed Applications*. 1998, Technion - Israel Institute of Technology. p. 163–173.
3. Holder, O., I. Ben-Shaul, and H. Gazit, *Dynamic Layout of Distributed Applications in FarGo*, in *21st Int'l Conf. Software Engineering (ICSE'99)*. 1999: ACM Press.
4. Moyer, S. and A. Umar, *The Impact of Network Convergence on Telecommunications Software*. IEEE Communications, 2001. January: p. 78–84.

Trading Freshness for Performance in a Cluster of Replicated Databases

Cécile Le Pape¹, Stéphane Gançarski¹, and Patrick Valduriez²

¹ Laboratoire d'Informatique de Paris 6, France

² INRIA/IRIN, Nantes, France

1 Introduction

In the Application Service Provider (ASP) model, applications and databases are hosted at the provider site and are accessed by customers, typically through the Internet, who are no longer concerned with data and application maintenance tasks. Through replication of customers' databases at several nodes, a cluster system can yield high-availability and high-performance at low cost. Our objective is to demonstrate the viability of the ASP model using a cluster system for pharmacy applications in France. In particular, we must support mixed workloads composed of front-office update-intensive transactions (e.g. drug sales) and back-office read-intensive queries (e.g. statistics on drugs sold). Our approach is to capture application semantics for optimizing load balancing within the cluster system. We use a mono-master replication scheme: updating transaction are all sent to a single master node while read-only queries may be sent to any node. Slave nodes are updated asynchronously through *refresh transactions* and consistency reduces to *freshness*. Application semantics mainly consist of a *freshness level*, which express how much the result of a given query may differ from the expected results if the query was sent to the master node.

2 Freshness Model

Users specify execution units for queries called *tries*. Each try contains both a minimal *freshness level* for the results of the query and a maximal execution *delay*. Tries are organized as a directed acyclic graph called *freshness policy*: when a try fails, i.e. if the results cannot be retrieved at the required freshness level within the specified delay, a next try is issued according to the partial order of the freshness policy. The *freshness level* of a query is defined as a logical *formula*: the results of the query at a slave node are fresh enough if the formula is satisfied at the node. A freshness level is composed of logical atoms, called *freshness atoms*. A freshness atom ($\mu(a) < t$) is a condition upon an *access atom* (relation, tuple, column or tuple attribute -also called element-) which bounds the staleness of the access atom a under a certain threshold t for a given *freshness measure* μ . Let a be an access atom; a_i is the secondary copy of a at a slave node S_i and $\hat{a}$ is the fresh state of a at the master node, we consider the following measures for tuple attribute access atoms (due to space limitations,

other access atoms are omitted here): **Numerical measure**: the freshness of a_i is the difference between a_i and $\hat{a}$: $Num_{elt}(a_i) = \hat{a} - a_i$. **Age measure**: the age of a_i is the time since a_i has been waiting for the first refresh transaction, t_k . **Ordering measure**: the ordering measure of a_i is the number of refresh transactions to apply to a_i to get $\hat{a}$.

3 Trading Freshness for Load Balancing

When choosing an execution node, the load balancer needs to know for every slave node if the freshness level is already satisfied by the node and which refresh transactions must be sent to the node if it is not fresh enough. To this end, we estimate a confidence level for each freshness atom ($\mu(a) < t$) and each slave node S_i , by computing the effects on a_i of the refresh transactions waiting to be applied on S_i . In this way, the load balancer knows, for each node, how many refresh transactions must be sent to reach the required freshness of a given query try. This information is used together with usual load information (node CPU load, network link load, ...) to determine the best node to perform a query try, with forced refresh if the node is not fresh enough. If the result of the query try is not given by the local DBMS within the required delay, a next try is then considered, at a possibly different node.

4 Implementation and Experimental Results

The prototype is implemented in Java. Data is replicated at four nodes, each running the Oracle 8i server under Linux. The coordinator (load balancer and freshness evaluator) runs on a separate node. All nodes are interconnected by a switched 1 GBit/s Fast-Ethernet LAN. First experiments were led with a mixed workload of SQL updates and SQL sums with grouping. First results are promising since they show that, if users accept to read *almost fresh* data (e.g. data stale since 30 seconds which is quite reasonable for most OLAP queries, particularly in pharmacy applications where statistics are computed at most daily) they may gain more than 50 % in query response time.

5 Future Work

We plan to perform more extensive experimentation, by varying the type and the frequency of read-only queries and the size of the workloads. We also want to study the impact on performances induced the level of granularity of access atoms, to see if it is worthy to define freshness policies at a very fine grain. Another crucial point is to measure the overhead induced by routing OLAP queries, evaluating freshness and performing refresh transactions. This will allow us to understand how our approach scales up with the size of the workload and the number of nodes.

Regarding the Integration of Collaborative Applications into Virtual Worlds

Roberta Lima Gomes^{1,2}, Guillermo J. Hoyos-Rivera^{1,3}, and Jean Pierre Courtiat¹

¹ LAAS-CNRS 7, Av. du Colonel Roche, 31077 Toulouse, France
{rgomes, ghoyos, courtiat}@laas.fr

² Researcher supported by a scholarship from CNPq, Brazil

³ Researcher of the Universidad Veracruzana. CONACyT scholarship 70360 and PROMEP

Abstract. In order to respond to the increasing demand for collaboration technologies, several Collaborative Virtual Environments have been proposed. However, most of them work independently of other collaboration tools, providing collaboration only inside their own shared virtual world. This work proposes the definition of a collaborative integration environment implementing a general framework for the integration of different collaborative applications with a conventional CVE. This integration aims to allow the collaboration activity to take place in several contexts in a parallel and coordinated way.

1 Introduction

The *Collaborative Virtual Environment* (CVE) systems [1] represent an important category of CSCW systems that provides collaboration facilities through the implementation of distributed and shared virtual spaces. Generally, these shared spaces are represented as rich 3D scenes, allowing users to be "immersed" in a virtual reality (VR) world where they accomplish several collaboration tasks. However, the complexity associated with the implementation of these environments has led the current CVE systems to promote proprietary solutions, working independently of other collaboration tools and providing collaboration only inside their own VR world.

This work proposes the development of a *Collaborative Integration Environment* (CIE) based on a general framework that allows CVE systems to be integrated with different collaboration tools. This environment aims to extend conventional CVEs in order to take the collaboration beyond the VR context. Collaboration activity can then take place in several contexts in a parallel and coordinated way.

2 The Collaborative Integration Environment

To accomplish the integration of originally independent collaboration tools, we have defined a general integration framework where distributed *Collaborative Applications* are to be connected through an *Integration Module* (Fig.1). Basically, this module interfaces with these applications listening to the events notified by them (*EventMessages*) and asking them to execute some actions (*ActionMessages*).

During a collaboration activity, each Collaborative Application is in charge of notifying all the relevant events it manages to the Integration Module (sending it

EventMessages). Once this last receives an *EventMessage*, it asks its *Configuration Module* to verify if there are any actions associated to this event. If so, the *ActionMessage Factory* creates the respective *ActionMessages* which are then sent to the target Collaborative Applications.

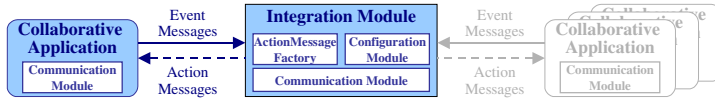


Fig. 1. General Integration Framework

Therefore, the Integration Module allows the association of events from one collaboration context with the execution of actions within another collaboration context. This association can be accomplished defining a list containing all the *ActiveEvents* (events that have associated actions) for each Collaborative Application. Once a Collaborative Application connects to the Integration Module, this last sends it the list of *ActiveEvents* defined to this application. This allows the Collaborative Application to know which are the relevant events (taking place in the collaboration activity it manages) that might be notified to the Integration Module.

Based on the defined framework we have implemented a CIE where different collaborative applications can be integrated. Platform independence and extensibility were our basic requirements. The first one is due to the heterogeneity of distributed systems. The second one aims to guarantee that the environment will not be restricted to the collaborative applications initially integrated. In order to fulfill these requirements, Java has been chosen as the underlying technology.

To provide different collaboration functionalities, we have chosen three collaborative applications to be integrated: VNet [2], as the CVE system, *CoLab* [3], a collaborative browsing system, and *Platine* [4], a platform providing communication (chat, audio and videoconference) and collaboration tools (shared whiteboard and application sharing tool).

In order to ensure the environments extensibility, a simple interfacing mechanism has been defined so that any collaborative application can be integrated. Just few integration steps are to be performed. The environment's flexibility is achieved through a configuration file (defining the list of all *ActiveEvents*) used to initially configure the Integration Module. To specify this file we have defined a new XML-based language called *Integration Module Configuration Language* (IMCL).

References

1. Churchill, E.F., Snowdon, D.N., Munro, A.J.: Collaborative Virtual Environments : Digital Places and Spaces for Interaction. Springer, London (2001)
2. Vnet Web Site: <http://www.csclub.uwaterloo.ca/u/sfwhite/vnet>
3. Hoyos-Rivera G.J., Gomes R.L., Courtiat, J.P.: A Flexible Architecture for Collaborative Browsing. In: WETICE'02, IEEE, USA Pittsburgh (2002)
4. Baudin V. *et al.*: Supporting distributed experts in e-meetings for synchronous collaboration. In: SMC'02, IEEE, Tunisia Hammamet (2002)

Workflow Reconfiguration Using Chains

Avigdor Gal, Eugeny Michailovsky, and Mati Golani*

Technion, Israel Institute of Technology

Workflows have become a common tool for modeling organizational activities. Research and practice of Workflow Management Systems (WfMS) involve ontological aspects (*e.g.*, the relevant constructs for modeling inter-organizational workflows [2,1]), design aspects (*e.g.*, synchronization of concurrent workflow instances [4]), reverse engineering [3], and optimization. The latter has seen few advances, especially when it comes to dynamic workflow design, where organizations need to continuously revise their workflows. As an illustrative example, consider virtual enterprises, enterprises that are formed in a competitive market to respond to the need to improve cost/performance ratio by cross-organizational cooperation. Cross-organizational operations require a dynamic flexible mechanism that can handle the data flow among different partners [5,6]. Therefore, if one partner delivers raw material to another partner for further processing, then whenever one partner changes the characteristics of the raw material (either a temporary or a permanent change) it should be reflected in the business process of the other (*e.g.*, machinery replacement). In environments that require fast response, an off-line update procedure decreases the efficiency of the organization and may cause unnecessary delays due to the need to restart the workflow engine with any new change. Therefore, an optimized process, involving online reconfiguration and flexible resource allocation, becomes an important feature of workflow management.

This work focuses on workflow optimization through task redesign. We have designed an algorithm for chain creation in workflows. A *chain* is a sequence of activities in a given workflow with no forks and joins. The proposed algorithm provides chains that maximize a measure of importance, combined from the priority a user assigns to a specific workflow outcome and the likelihood of such outcome, based on historical data. The use of chains is beneficial for several reasons. First, chain-based workflow design can improve process locality. Second, the generation of a chain increases local task management and reduces the role of a centralized workflow management system, thus improving system resiliency. As an example, consider the ability to affect customer satisfaction using workflow redesign. Customer satisfaction can be measured in multiple methods, from an explicit score a customer assigns to a completed transaction to implicit means such as wait-time and number of mid-process aborts. For the sake of simplification, suppose that the customer satisfaction is derived from the total processing time, and most orders are submitted by non-preferred customers. Since item query and purchase activity are independent activities, and the flow of data among the processes consumes bandwidth, lengthy processing time may incur.

* A member of the Active Management Technologies group at IBM Research Laboratory in Haifa.

Dynamic workflow system, geared towards customer satisfaction, should identify reduction in customer satisfaction and switch to a mode in which both processes are localized to reduce bandwidth consumption, sometime at the expense of more privileged customers. The proposed algorithm is designed to provide rapid response to continuously changing workflows, in that its polynomial performance allows online optimization.

References

1. E. Colombo, C. Francalanci, and B. Pernici. Modeling coordination and control in cross-organizational workflows. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002 Irvine, California, USA, October 30 - November 1, 2002, Proceedings*, volume 2519 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2002.
2. A. Gal and D. Montes. Inter-enterprise workflow management systems. In *Proc. 10th International Workshop on Database & Expert Systems Applications*, pages 623–627, Florence, Italy, September 1999.
3. M. Golani and S.S. Pinter. Generating a process model from a process audit log. In M. Weske W. van der Aalst, A. ter Hofstede, editor, *Lecture Notes on Computer Science, 2678*, pages 136–151. Springer Verlag, 2003. Proceedings of the Business Process Management International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003.
4. C. Heinlein. Synchronization of concurrent workflows using interaction expressions and coordination protocols. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002 Irvine, California, USA, October 30 - November 1, 2002, Proceedings*, volume 2519 of *Lecture Notes in Computer Science*, pages 54–71. Springer, 2002.
5. L. Song and R. Nagi. An integrated information framework for agile manufacturing. In *Industrial Engineering Research Conference proc. IIE*, pages 568–573, Norcross, GA, 1996.
6. L. Song and R. Nagi. Design and implementation of virtual information system for agile manufacturing. *IIE Transactions*, 29(10):839–857, 1997.

Modeling Semistructured Data by Using Graph-Based Constraints

Ernesto Damiani¹, Barbara Oliboni², Elisa Quintarelli³, and Letizia Tanca³

¹ Dipartimento di Tecnologie dell'Informazione – Università degli Studi di Milano
damiani@dti.unimi.it

² Dipartimento di Informatica – Università degli Studi di Verona
oliboni@sci.univr.it

³ Dipartimento di Elettronica e Informazione – Politecnico di Milano
{quintare,tanca}@elet.polimi.it

The widespread adoption of semistructured and more in particular XML data has opened a new venue for the application of formal methods to the specification of models for defining classes of semistructured or XML documents [1,3]. Most current models for semistructured data are too specific to allow immediate comparison with each other, and do not easily support incremental model design. For this reason, we introduce the *General Semistructured Meta-Model* (GSMM), a simple meta-model for semistructured information which can be applied for the translation to a common formalism of the various abstract models proposed in literature; this approach fosters easy *a priori* comparison and discussion of concrete models' features, such as allowed sets of values, handling of object identifiers, relationships representation; moreover, it supports effective inter-model translation and design. Unlike other proposals [2], our meta-model does not rely on an extension of the concept of schema; rather, GSMM is based on a *generic* graph that can be instantiated into a number of *concrete* models by providing a) values for a restricted set of parameters (labels) and b) some high-level constraints, themselves represented as graphs. Here, constraints take the role of the schema, imposing *constraints* directly on the structure of semistructured data provides the designer with a powerful tool for controlling the degree of precision of the structure and introducing flexibility at the data representation level.

As an example, we now show how to represent the XML standard data model, called *Infoset* [3] by using GSMM. Infoset represents both XML schemata and documents as *multi-sorted trees*, i.e. trees including nodes (called *information items*) belonging to a variety of types. In other words, an XML document's information set consists of a number of information items; the information set for any well-formed XML document will contain at least one document information item (the root of the tree) and several others. An information item is an abstract description of some parts of an XML document: each information item has a set of associated named *properties*

An *XML Infoset graph* is a *semistructured rooted data graph* $\langle N, E, r \rangle$, where: a) the cardinality $|NL|$ of the sets of node labels is 6. Each node n_i has as tuple of labels $NL_i = \langle Ntag_i, Ntype_i, Norder_i, Ncontent_i, URI_i, Namespace_i \rangle$; the type label $Ntype_i$ indicates whether the node is the root, an element, text, attribute, processing instruction or comment, whereas the label $Norder_i$ assumes as value a

natural number representing the relative order of the node w.r.t. other children of its parent node, or $\perp$ (undefined) for root, text and attribute nodes. Moreover, the label $Ncontent_i$ can assume as value a PCDATA or $\perp$ (undefined). The two labels URI_i , representing the resource identifier attached to that node, and $Namespace_i$, representing the node namespace. b) The cardinality $|EL|$ of the set of edge labels is 1, where the unique label represents the edge type. Each edge $e_j = \langle (n_h, n_k), EL_j \rangle$, with n_h and n_k in N , has a label $EL_j = \langle Etype_j \rangle$, where the label $Etype_j \in \{\text{sub-attribute of}, \text{sub-element of}\}$. Note that edges represent the “containment” relationship between different items of a XML document, thus edges do not have names.

$r \in N$ is the root of the graph. Moreover, we add a high-level constraint (see Fig. 1) that imposes the edge type on the basis of the node type. In fact, in a XML document “sub-element of” edges point to “element” and originate from “element” nodes, whereas “sub-attribute of” edges originate from “element” and point to “attribute” nodes. In the InfoSet terminology, this is informally stated by requiring that the **children** property of an element item contains either elements or attributes, while attribute items cannot have a children property.

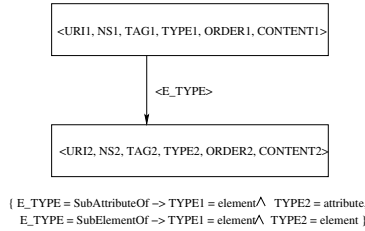


Fig. 1. In the XML InfoSet the edge type depends on the type of the endpoint node.

The graphical constraint in Figure 1 is expressed by using the *General Semistructured Language* (GSL), a formalism capable to express queries as well as (traditional and high level) constraints on semistructured data in a concise and unambiguous way. Like our meta-model, GSL relies on a terse graphical formalism offering high expressive and modeling power and easily understandable by non-specialists: in general, a constraint is composed by a *graph*, which is used to identify the subgraphs (i.e. the portions of a semistructured database) where the constraint is to be applied, and a set of *formulae*, which represent restrictions imposed on those subgraphs.

References

1. S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of the International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 262–275, 1997.
2. D. Beech, A. Malhotra, and M. Rys. A formal data model and algebra for xml, 1999.
3. World Wide Web Consortium. XML Information Set, 2001.
<http://www.w3c.org/xml-infoset/>.

A Graph-Based Model for Semistructured Temporal Data

Carlo Combi¹, Barbara Oliboni¹, and Elisa Quintarelli²

¹ Dipartimento di Informatica, Università degli Studi di Verona
Ca' Vignal 2 – Strada le Grazie 15 – 37134 Verona (Italy)
{combi,oliboni}@sci.univr.it

² Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 — 20133 Milano (Italy)
quintare@elet.polimi.it

Abstract. We sketch here the semistructured temporal data model GEM (Graphical sEmistructured teMporal), which is a graph-based data model and allows one to uniformly capture different temporal aspects of semistructured data, such as valid and transaction times.

1 Representing Semistructured Temporal Data

Modeling semistructured temporal data requires a complete formalization of the constraints related to the considered temporal dimensions, as it happened in the temporal database area for temporal relational data [2].

In the following, we sketch some main features of a graphical temporal data model (GEM: Graphical sEmistructured teMporal) general enough to represent both valid and transaction times: the *valid time* (VT) of a fact is the time when the fact is true in the modeled reality, whereas the *transaction time* (TT) of a fact is the time when the fact is current in the database and may be retrieved [2]. Valid time is usually provided by database users, while transaction time is system-generated and supplied. To represent semistructured temporal data we consider rooted, connected, directed, labeled graphs, where the temporal dimension is explicitly reported on node and edge labels and is described by means of an interval. We have two kinds of nodes: *complex* and *simple* nodes. The former represent abstract entities, whereas the latter represent primitive values.

Focusing mainly on TT, let us now consider how to manage it in a GEM graph: by TT we are able to keep trace of changes in the graph due to node and edge insertions, deletions, and updates. In this perspective, a GEM graph can be seen as a representation of the changes of a (atemporal) graph. Thus a GEM graph considering TT represents a collection of several *atemporal graphs*, each of them being the result of some operations on the previous one. The current graph is composed by nodes and edges having the transaction time ending with the special value *now*. The main idea is that operations on a graph must guarantee that every action on a node or on an edge still leaves a rooted, connected graph. In other words, the (current) nodes and edges must compose a rooted connected

graph. In our proposal changes are timestamped by TT, thus we can say that the overall GEM graph represents the graph history, which tells us the sequence of intermediate graphs up to the current one. Temporal labels (timestamps) of nodes and edges must satisfy some constraints, which guarantee that after each operation we still have a (atemporal) graph. As an example, the time interval of an edge between nodes must be related to the time interval of the related nodes. Intuitively, a relation between two nodes cannot be established before that both nodes are present in the graph and cannot survive the deletion of one of the two connected nodes. Figure 1 depicts an example of this constraint: part a) shows two generic nodes and the connecting edge, whereas part b) shows an example of time intervals of the nodes and of the edge that satisfy the constraint.

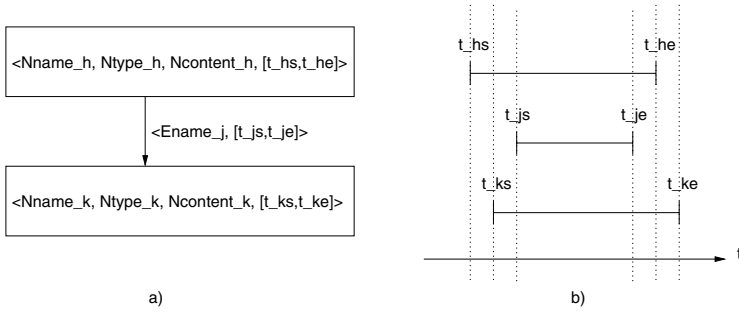


Fig. 1. The TT constraint on the time interval of a generic edge.

Thus, each operation on a GEM graph corresponds to the suitable management of temporal labels of (possibly) several nodes and edges: after each operation we have a consistent GEM graph. It is worth noting that we avoid the problem described and managed in [1] of having incorrect sequences of operations at a given time: any sequence of the following operations is correct, being composed by single operations, each of them producing a GEM graph.

When considering VT, a GEM graph represents the history of the given application domain: constraints and operations must guarantee that the history is consistent. As an example, it is not possible that an abstract entity has different values for the same property at the same time instant. It is important to observe that, in this case, the GEM graph cannot be considered the representation of a set of (atemporal) graphs, each of them being composed by nodes and edges valid at a given time instant.

References

1. S. S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *Theory and Practice of Object Systems*, 5(3):143–162, 1999.
2. C. S. Jensen and R. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–44, 1999.

A Formal Approach to Knowledge Base Comparison and Interoperability

Dan Corbett

Advanced Computing Research Centre
School of Computer and Information Science
University of South Australia
Adelaide, South Australia 5095

1 A Formal Definition of Ontology

This section draws on previous work on formal definitions for knowledge representation as defined by Conceptual Graph Theory [1–3]. We first define a background universe for our ontologies to exist in, which will give substance and order to the ontologies. The canon defines all the individuals that can exist, all possible relations between the individuals, and also imposes an ordering on the types of individuals and relations.

Definition 1. Canon. A canon is a tuple $(T, I, \leq, ::, B)$ where

T is the set of types. We will further assume that T contains two disjunctive subsets T_C and T_R containing types for concepts and relations.

I is the set of individuals.

$\leq \subseteq T \times T$ is the subtype relation.

$:: \subset I \times T$ is the conformity relation.

B is the Canonical Basis function.

This definition of canon provides the background for the representation, including the definitions of relation, subsumption and consistency to support our definition of an ontology. We can now formally define an ontology as the particular set of hierarchies that are created for a given domain, along with all of the operations on a canon.

Definition 2. Ontology. An ontology in a given domain M with respect to a canon is a tuple (T_{CM}, T_{RM}, I_M) where

T_{CM} is the set of concept types for the domain M and T_{RM} is the set of relation types for the domain M .

I_M is the set of individuals for the domain M .

An ontology is a collection of types and individuals, which forms a framework for the knowledge in a domain, arranged into a hierarchy based on the subtype relation $\leq$. The ontology (as a concept type hierarchy) acts as the framework, with conceptual graphs that conform to the hierarchy used to instantiate concepts in the domain.

2 Projection of Ontology Types, and Merging Knowledge

The following definitions of projection are modified from the standard definition used in recent Conceptual Graph literature [1, 3]. These definitions represent projection of types and define the subsumption operator on type hierarchies and thereby ontologies.

Definition 3. Concept projection. Given two concept types, s and t , s is said to have a projection into t if and only if there is a morphism $h_C: C \rightarrow C'$, such that:

$\forall c \in s$ and $\forall c' \in t'$, $h_C(c) = c'$ only if $type(c) \geq type(c')$, and $referent(c) = *$ or $referent(c) = referent(c')$

C is the set of concepts, $type: C \rightarrow T$ indicates the type of a concept, and $referent: C \rightarrow I$ indicates the referent marker of a concept.

Definition 4. Relation projection. Given two relation types, s and t , s is said to have a projection into t if and only if there is a morphism $h_R: R \rightarrow R'$, such that:

$\forall r \in R$ and $\forall r' \in R'$, $h_R(r) = r'$ only if $type(r) \geq type(r')$

R is the set of relations, and $type: R \rightarrow T$ indicates the type of a relation.

Given the canonical formation rules for conceptual graphs, we now have a complete formalism for the specification and manipulation of knowledge bases expressed using ontologies. If an ontology author is careful to set up the ontology according to the definitions of subsumption and projection, and then follows the canonical formation rules when deriving new types, expressions and graphs, the result will always be graphs and extensions to the ontology which are canonical according to the domain.

As an operator for ontology filtering and comparison, the use of the projection operator becomes obvious. A user would implement an ontology as a type hierarchy locally. Whenever the user sends a query to a database or knowledge base, or performs a web search, the user's ontology is compared with the ontology which has (hopefully) been implemented by the designer of that knowledge base. The query is examined in light of the local ontology and the remote ontology, and information from the remote knowledge base is filtered and constrained based on the comparison.

This technique can be used for two practical purposes: comparing and merging ontologies, or to inform and filter queries. Inheritance hierarchies can be seen as lattices that admit unification and generalization [4]. So, in our case, comparing and merging two ontologies is the process of finding the common points in the two ontologies (represented as lattices) and merging the rest of the structures together, in a similar manner to the unification of two CGs.

References

1. Mugnier, M.-L. and M. Chein, "Représenter des Connaissances et Raisonner avec des Graphes". *Revue d'Intelligence Artificielle*, 1996, **10**(6): p. 7–56.
2. Sowa, J.F., *Conceptual Structures: Information Processing in Mind and Machine*. 1984, Reading, Mass: Addison-Wesley.
3. Corbett, D.R. "Reasoning with Ontologies by Using Knowledge Conjunction in Conceptual Graphs". in *Proc. International Conference on Ontologies, Databases and Applications of Semantics*. 2002. Irvine, California, USA: Springer. Published as LNCS volume 2419.
4. Knight, K., "Unification: A Multidisciplinary Survey". *ACM Computing Surveys*, 1989, **21**(1): p. 93–124.

COVA: A Web-Based Learning System for Content-Based Lecture Retrieval

Guang-Ho Cha*

Department of Multimedia Science, Sookmyung Women's University
Seoul 140-742, South Korea
ghcha@sookmyung.ac.kr

Abstract. This paper introduces a prototype system called COVA that enables remote users to access specific parts of interest by contents from a large lecture database. COVA includes several novel techniques to achieve the content-based lecture retrieval in distance learning: (1) The semistructured XML model; (2) The technique to build structural summaries of XML lecture databases; (3) Indexing methods to speed up the search to find appropriate lecture contents.

1 System Architecture of COVA

We are currently developing a system for distance learning called COVA (Content-based Video Access) within our CyberUniversity project. The system includes seven major components: text processing and annotation, video processing and annotation, structural database summarizer, indexing methods, storage manager, browsing/query processing, and streaming media delivery (see Fig. 1).

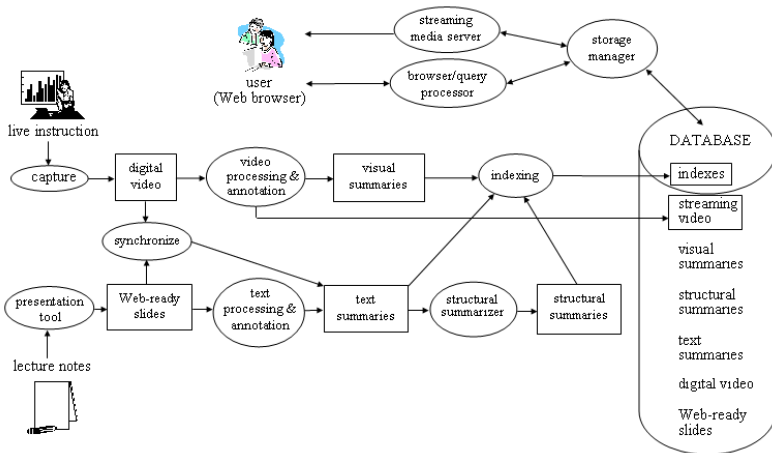


Fig. 1. COVA system architecture

* This Research was supported by the Sookmyung Women's University Research Grants 2003.

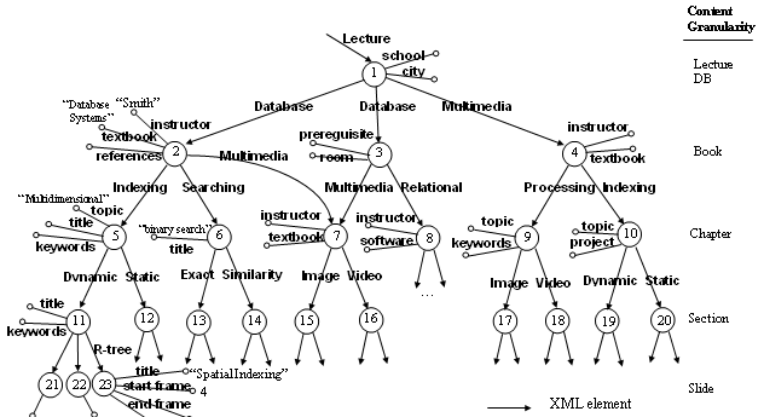


Fig. 2. An example lecture database

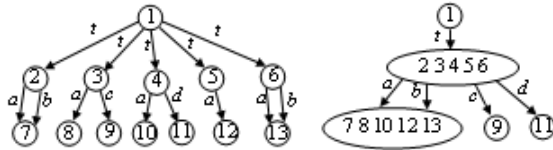


Fig. 3. COVA schema extraction: (a) a database graph (b) the extracted COVA schema

2 Data Model, Schema, and Indexing Methods

We adopt the XML-based semistructured model to describe the lecture content. Thus a lecture database is thought of as a labeled directed graph. Fig. 2 depicts a portion of a lecture database containing three class lectures. The motivation to employ the semistructured model comes from the need to provide the lecture content description with flexibility and diversity. COVA also has a database summarizer that builds a schema from lecture databases (see an example in Fig. 3). It provides a compact structural summary of lecture databases. COVA currently includes two indexing methods, the *P-index* (path index) for path indexing on a database graph and the *GB-index* (grid bitmap index) [1] for indexing images in the lecture content.

3 Conclusion

We presented a new approach to facilitate the retrieval of specific parts of lectures by content. Some novel techniques to achieve content-based lecture retrieval were successfully incorporated in our distance learning system COVA. We believe that COVA will provide a valuable education tool for remote and future users.

References

1. G.-H. Cha, Bitmap Indexing Method for Complex Similarity Queries with Relevance Feedback, Proc. ACM Workshop on Multimedia Database (2003) to appear.

An Ontology for Semantic Middleware: Extending DAML-S Beyond Web-Services

Daniel Oberle¹, Marta Sabou², Debbie Richards³, and Raphael Volz¹

¹ Institute AIFB, University of Karlsruhe, Germany
`lastname@aifb.uni-karlsruhe.de`

² Department of Artificial Intelligence, VU Amsterdam, The Netherlands
`marta@cs.vu.nl`

³ Computing Department, Macquarie University, Sydney, Australia
`richards@ics.mq.edu.au`

1 Introduction

Describing software entities using Semantic Web technology is a growing research area. Our work investigates the semantic description of software entities that provide an application programmer's interface (API) to allow reasoning with the descriptions and their interrelationships. We present an ontology for our semantic middleware, called Application Server for the Semantic Web [3], where it is used to facilitate implementation tasks and semantic discovery. Building on an emerging standard from the Semantic Web community, our work includes a number of extensions to DAML-S [1] which currently allows semantic description of a particular type of software entities, viz. web-services, in order to facilitate their automatic discovery and integration.

2 The Ontology

Although DAML-S serves as a good starting point for our ontology, the main difficulty was in the type of software entities to be described. While DAML-S describes web-services, our goal is to describe software modules' APIs along other properties. As a result some parts of DAML-S were not reusable. Figure 1 presents the main ontologies in DAML-S in comparison with the ontologies we have developed. The following discussion is organised using our design principles.

1. Semantic vs. Syntactic descriptions. We have adopted the separation between semantic and syntactic descriptions in order to achieve a flexible mapping. A number of our ontologies allow semantic description and others are used for syntactic descriptions. A mapping exists between the description of both aspects. However, given the different type of entities we want to describe, we modified some of the DAML-S ontologies as follows:

- we have kept the DAML-S *Profile* ontology for specifying semantic information about the described modules and extended it with a few concepts for describing APIs at the conceptual level which are grouped in a small ontology called *API Description*.

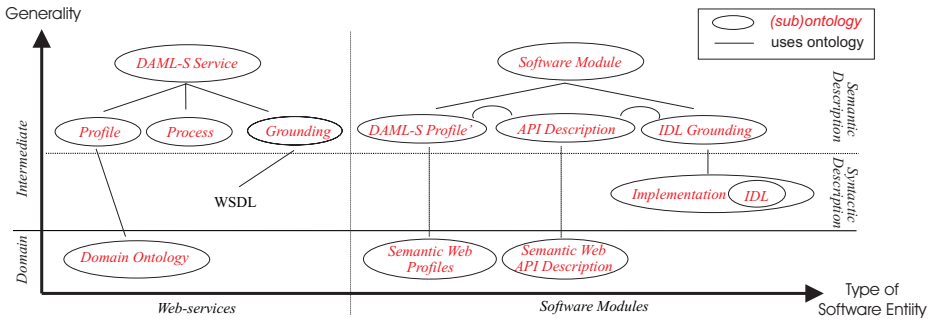


Fig. 1. Ontology overview

- we did not use the *Process* ontology since we are not interested in the internal working of the modules.
- we formalized a subset of *IDL* terms (Interface Description Language) and use them to describe the syntactic aspects of APIs in an *Implementation* ontology.
- as a consequence of the changes above, we could not reuse the existing DAML-S *Grounding* and wrote an *IDL Grounding* ontology.

2. *Generic vs. Domain knowledge.* Our core ontology allows specifying semantic and syntactic knowledge about APIs in a generic way facilitating its combination with domain knowledge. For our semantic middleware we have built two domain ontologies in the area of the Semantic Web. The *Semantic Web Profiles* ontology specifies the type of existent Semantic Web software modules at a very coarse level. *Semantic Web API Description* describes the functionality of APIs at a more fine grained level. Our approach can be described in terms of the ONIONS [2] ontology development methodology which advises grouping knowledge with different generality in generic, intermediate and domain ontologies.

3. *Modularity.* Modularity enables easy reuse of specifications and extensibility of the ontology. We apply this principle by making an effort to centralize related content to a certain concept whose instance can be reused at description time. We decided to group together chunks of information that are most likely to be reused. Like shown in Figure 1 we have grouped this information in small ontologies which are used by others.

References

1. M. H. B. et al. DAML-S: Web service description for the Semantic Web. In *Proc. of ISWC 2002*, volume 2342 of *LNCS*, pages 348–363. Springer, 2002.
2. A. Gangemi, G. Steve, and F. Giacomelli. ONIONS: An ontological methodology for taxonomic knowledge integration. In *Proc. of ECAI-96*, 1996.
3. D. Oberle, R. Volz, B. Motik, and S. Staab. *An extensible open software environment*. International Handbooks on Information Systems. Springer, 2003.

A Formalism of Contextualized Predicate Calculus for Information Integration

Jijuan Zheng

STARLab, Vrije Universiteit Brussel
Zheng.jijuan@vub.ac.be

Abstract. This paper introduces an extension to predicate calculus, namely *contextualized predicate calculus* (C-PC) within the DOGMA ontology framework [1]. We briefly present both syntax and semantics of contextualized predicate calculus, together with its use in information integration and querying.

1 Contextualized Conceptualization

A conceptualization C is usually defined as a structure $\langle D, R \rangle$, where D is a domain consisting of ordinary objects and R is a set of relevant ordinary relations on D . We extend the traditional predicate calculus with an extension of notion of context to handle multiple conceptualizations and possible operations between them.

In our extended predicate calculus, a context is represented as a special object called *context object*, and every formal thing is always defined within a given context. We use notation $\gamma.x$ to indicate that x is defined according to the context γ , to restrict x within context γ . The prefix ' γ .' can go as far as necessary, e.g., $\gamma_n. \dots \gamma_1.x$ depending on how many layers of contexts across between the context to which x is defined and the current context. We call the process of limiting a formal entity within a certain context *contextualization*. A conceptualization based on such formalism is called a *contextualized conceptualization*.

2 Syntax of C-PC

We define a contextualized predicate language as (A, W) as in predicate language with additional *context constants*.

Some more basic extensions are:

A *context constants sequence* is a form of $v_n . \dots .v_2 .v_1$ where v_i are context constants only.

A variable, a constant, a context constant of A , prefixed with a *context constants sequence* is a term. E.g., $v_n . \dots .v_2 .v_1 . \tau$ is a term.

If $\mu.P$ is an n -ary predicate of A and $v_1.t_1, \dots, v_n.t_n$ are terms, then $\mu.P(v_1.t_1, \dots, v_n.t_n)$ is an *atomic formula*. $\mu.P(t_1, \dots, t_n)$ is a *ground atomic formula* iff $v_1.t_1, \dots, v_n.t_n$ are all

constants. An atomic formula is a *well formed formulae*. If $\gamma_1.W_1$ and $\gamma_2.W_2$ are *wffs*, then $(\gamma_1.W_1 \wedge \gamma_2.W_2)$, $(\gamma_1.W_1 \vee \gamma_2.W_2)$, $(\gamma_1.W_1 \Rightarrow \gamma_2.W_2)$, $(\gamma_1.W_1 \Leftrightarrow \gamma_2.W_2)$, $\neg\gamma_1.W_1$ are also *wffs*. If $v_1.x$ is a variable and $v_2.W$ is a *wff*, then $\forall(v_1.x) \bullet v_2.W$ and $\exists(v_1.x) \bullet (v_2.W)$ are *wffs*. Here $\exists$ is existential quantifier, and $\forall$ is universal quantifier.

3 Semantics of Contextualized Predicate Calculus

Given a conceptualization $\gamma C = (\gamma D, \gamma F, \gamma R, \gamma I)$ and a contextualized predicate language (A, W) , an interpretation I is a mapping from constants of the language to elements of the conceptualization within context γ .

A sentence $v.\phi$ is satisfied by an interpretation and a variable assignment U in context γ , with formal notation $\gamma \models_I v.\phi [U]$. An example: $\gamma \models_I v.p(v_1.x, \dots, v_n.x_n)[U]$ iff $\langle T_{IW}(v_1.x_1), \dots, T_{IW}(v_n.x_n) \rangle \in I(v.p)$.

4 Use of C-PC for Integration and Querying

To integrate $\beta_1.C_1$ and $\beta_2.C_2$ in a new context γ , we define the integrated conceptualization as $\gamma.C'$. A merged conceptualization $\gamma.C' = \gamma.((\beta_1.D_1 \cup \beta_2.D_2), (\beta_1.F_1 \cup \beta_2.F_2 \cup \gamma.A_1), (\beta_1.R_1 \cup \beta_2.R_2 \cup \gamma.A_2), (\beta_1.I_1 \cup \beta_2.I_2))$.

The set of all new predicates α_i make a new set $\gamma.A = \{\gamma.\alpha_1, \dots, \gamma.\alpha_n\}$, called *articulation predicates* here.

Most of current research concerning articulations is basically equivalency and subsumption relations between two conceptualizations, whereas the proposed contextualized predicate calculus enables the formulations of more sophisticate relations among different conceptualizations defined within different contexts.

A query for $L = (A, W)$ is any expression of the form $\langle \bar{v}.\bar{x} / \bar{v}.\bar{t} \mid W(\bar{v}.\bar{x}) \rangle$

A data source is formalized as $DS = (\gamma, L, I, IC)$, where γ is the *context* of DS . A set of data sources is called a Data Source Space $DSS = \{DS_i\}$.

Intuitively, the query $\langle \bar{v}.\bar{x} / \bar{v}.\bar{t} \mid W(\bar{v}.\bar{x}) \rangle$ over DSS is meant to return the set of all tuples of constants $\bar{v}.\bar{c} = v_1.c_1, \dots, v_n.c_n$ such that each $v_i.c_i$ satisfies the type $v_i.\tau_i$, and each of these tuples satisfies $W(\bar{v}.\bar{x})$ by each corresponding interpretation I_i within each corresponding context γ_i . In case $\langle \bar{v}.\bar{x} / \bar{v}.\bar{t} \mid W(\bar{v}.\bar{x}) \rangle$ is a *IC* of DSS , it means that, for all constants $\bar{v}.\bar{c} = v_1.c_1, \dots, v_n.c_n$ such that each $v_i.c_i$ satisfies the type $v_i.\tau_i$, and the all DS in DSS satisfies $W(\bar{v}.\bar{x})$.

Reference

1. Spyns, P., Meersman, R. & Jarrar, M.: Data modelling versus Ontology engineering. in Sheth A. & Meersman R. (ed.), SIGMOD Record Special Issue on Semantic Web, Database Management and Information Systems (2002)

An Ontology Based Visual Tool for Query Formulation Support

Tiziana Catarci¹, Tania Di Mascio², Enrico Franconi³, Giuseppe Santucci¹,
and Sergio Tessaris³

¹ University of Roma “La Sapienza”, Italy; {Catarci,Santucci}@dis.uniroma1.it

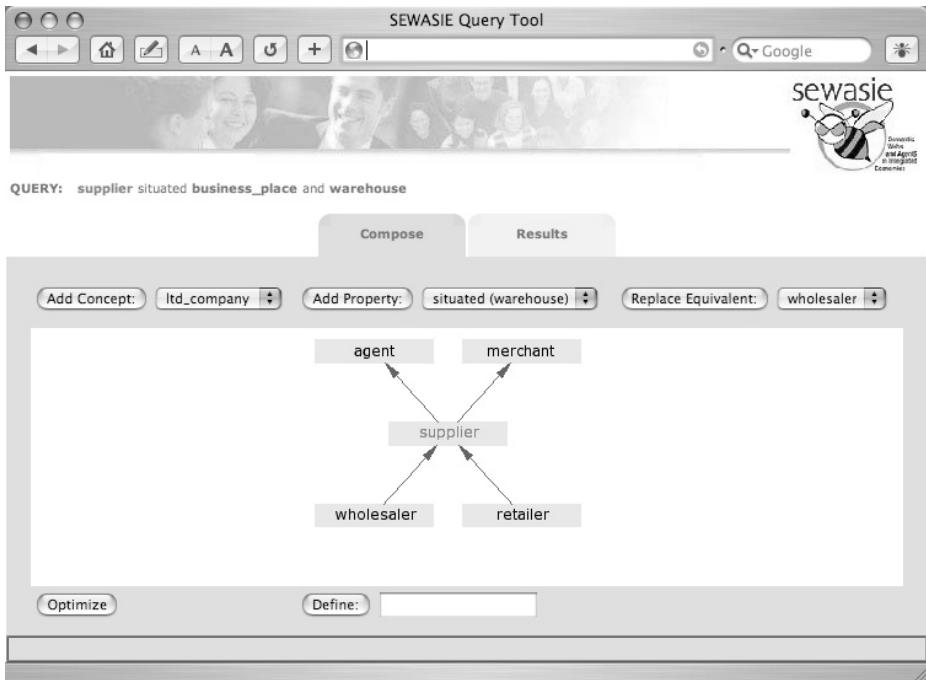
² University of L’Aquila, Italy; tania@ing.univaq.it

³ Free University of Bozen-Bolzano, Italy; {Franconi,Tessaris}@inf.unibz.it

Abstract. The SEWASIE (SEmantic Webs and AgentS in Integrated Economies) European IST project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The overall project development strictly follows a user-centred design methodology. Users have been involved from the very beginning and are constantly participating in the design and testing of the system, in order to obtain a first example of visual interface user-dedicated for an ontology-based search engine. In the poster we describe the end-user interface component for query composition. We describe the user-centred design of the interface, the underlying logic based technologies, and the formal principles of support and interaction.

In this poster we describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query (or a non nested Select-Project-Join SQL query) ready to be executed by some evaluation engine associated to the information system.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user’s rich vocabulary. The user can use the ontology’s vocabulary to formulate the query, and she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus solving the critical aspect that only very skilled users have available sufficient knowledge about the schema of the stored information and, therefore, are able to formulate significant queries.



Queries can be specified through an iterative refinement process supported by the ontology through intentional navigation. The user may specify her/his request using generic terms; after the query classification, which makes explicit the meaning with respect to the ontology and the specificity of the query itself and of the subterms composing the query, the user may refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, but by giving an explicit meaning to a query and to its subparts through classification.

The overall project development strictly follows a user-centred design methodology. Users have been involved from the very beginning and are constantly participating in the design and testing of the system, in order to obtain a first example of visual interface user-dedicated for an ontology-based search engine. In particular, they are giving key hints to build the user-interface, which is a crucial component of the overall SEWASIE architecture. So, in this poster we will give particular emphasis to the steps in user-centred design we have followed in this project.

In the poster the underlying technologies are introduced, from the point of view of the query expressiveness, the ontology support, and the natural language verbalisation issues. A fully worked out example and the related works are also presented.

Workshop PC Chairs' Message

The Industry Program Board of the Federated Conferences Event has solicited reports on relevant case studies, intended to cover industry practice related to the subjects treated by these conferences, which straddle the intersection of distributed computing, databases, and cooperative systems, as their respective technologies and methodologies increasingly become applied in the context of the internet and semantic web.

Evaluation of submissions was primarily based on relevance, general applicability and discussion potential for the solutions and novel approaches. These selected case study reports are being presented in a separate 1-day Industry Program Workshop targeted at audiences interested in practical experience, best practices and corporate strategy related to all aspects of distributed internet- and intranet-based computing. The participants thereby had a uniquely easy opportunity to access presentations in the scientific program, especially the shared keynote talks of all three conferences.

The federated conferences in this way aspire to become a crossroads event in combining distributed computing infrastructure, tools and applications in organizations.

August 2003

Robert Meersman, VU Brussels, Belgium
Michael Brodie, Verizon Information Technology, USA
(OTM'03 Industry Program Workshop 2003 Program Committee Co-chairs)

Revaluation of a Large-Scale Thesaurus for Multi-media Indexing: An Experience Report^{*}

Dirk Deridder¹ and Peter Soetens²

¹ Vrije Universiteit Brussel (VUB), Programming Technology Lab, Pleinlaan 2,
1050 Brussels, Belgium

Dirk.Deridder@vub.ac.be

<http://prog.vub.ac.be/>

² Vlaamse Radio- en Televisieomroep (VRT), Auguste Reyerslaan 52,
1043 Brussels, Belgium

peter.soetens@vrt.be

<http://www.vrt.be/>

Abstract. In this paper we provide a preliminary overview of a number of problems we encountered when faced with the revaluation of a large-scale mono-lingual thesaurus. The thesaurus we speak of is used to wade through the vast multimedia archive of the Flemish public radio and television broadcaster (VRT). In order to support advanced and ‘knowledgeable’ queries on the archive, it became imperative to upgrade the existing infrastructure. In this context we performed an in-depth analysis of the existing legacy situation. This led to the identification of a number of structural problems as well as problems with respect to content. Solutions to counter some of these have already been established. To support the new search-requirements for the archive, we have migrated the existing system to an ontology-inspired infrastructure.

1 Introduction

The work we present in this paper was performed in the context of the e-VRT MPEG project which consisted of a collaboration between VRT, VUB, and IMEC. The central theme was to investigate and develop the necessary technology to set up an enterprise-wide content management system in the context of a public radio and television broadcaster. In this paper we will focus on a number of experiences obtained in the work-package that concentrated on meta-data management for a multimedia archive. As input for this package we had access to an existing thesaurus that contained 229,860 lead terms. This collection of words continues to grow as we speak, since new entries are added on a regular basis by a team of highly skilled thesaurus administrators. It is a mono-lingual (Dutch) collection that is mainly used to wade through the vast multimedia archive (partly digital) of VRT. To support the archives’ search engine, an intermediate database system exists in which the lead term annotations of the

^{*} This work was funded by the Flemish government (Belgium).

archived items reside (over 500,000 digital archive documents with a growth of approximately 30,000 documents each year¹). At this moment almost 90% of the archive searches are based on these lead term annotations. Besides the trivial use of the archive as a reference work for newscasts, it is also consulted for the purpose of creating game shows and documentaries for instance. It is clear that the multimedia archive as well as the thesaurus are considered as main assets for the broadcaster. Both are used on a daily basis and are under constant pressure to accommodate new demanding search operations. This has led to usages of the thesaurus infrastructure in a way that was never (or could never have been) anticipated at the time of its conception (around 1986²). It shouldn't be surprising that this resulted in a number of 'inconsistencies' in the data as well as a number of creative abuses of the existing tools. Hence they have a major interest in investigating new ways of organizing and managing the archives' meta-data by reevaluating the existing thesaurus. To support advanced and more 'knowledgeable' queries we have opted for an ontology-inspired infrastructure. For this purpose we split up our activities in a content-wise and a structure-wise reevaluation process.

In this paper we provide a preliminary overview of a number of problems we encountered when faced with the reevaluation of such large-scale thesaurus legacy. In Section 2 we will zoom in on a number of results from our structural and statistical analysis of the thesaurus (content as well as infrastructure). Based on these results we will discuss the conversion and reevaluation of the thesaurus into an ontology-based infrastructure in Section 3. To conclude we will also present a number of elements we believe to be important for our future work.

2 Analysis of the Existing Thesaurus

2.1 Numerical Analysis

The thesaurus we analyzed contained 229,860 lead terms (LT). Between these lead terms we counted 147,245 relationships, which boils down to a ratio of 0.64 relationships per LT. At first sight this seems very low, but it is an expected result considering the small set of relationship types available in the thesaurus (broader term - BT, narrower term - NT, use for - UF, use - USE, related term - RT, scope note - SN). In Table 2.1 we present the 30 LTs that have the highest relationships/LT ratio and hence can be considered as main focal points from a numerical perspective. Since cross-language semantical ambiguities are not relevant in the work we present here, we have taken the liberty to translate the lead terms to English. Note that one of the main archive consumers is the newscast department. Consequently, the top level LTs in Table 2.1 clearly reflect

¹ This figure excludes the non-digital documents over a 30 year period. Due to resource restrictions, the digitization of these documents is currently put on hold.

² Even though the software has evolved since then, the underlying principles have not (i.e. a rigid thesaurus representation with a limited number of relationships to capture lead term semantics).

their main topics of interest (i.e. countries, sports teams, airline companies, newspapers, ...)

Table 1. Top 30 lead terms based on the number of relationship references

#	Lead Term	Ratio	#	Lead Term	Ratio
1	FRANCE	1.018	16	RIVER	278
2	USA	748	17	PLANT	277
3	BELGIUM	654	18	RUSSIAN FEDERATION	251
4	GERMAN SITES	592	19	REWARD	246
5	AUTHOR	519	20	NEWSPAPER	245
6	GREAT BRITAIN	488	21	SWITZERLAND	243
7	THE NETHERLANDS	477	22	COMPUTER COMPANY	216
8	ITALY	444	23	AUSTRIA	211
9	TELEVISION PROGRAMME	428	24	AIRLINE COMPANY	208
10	JOURNAL	368	25	DISEASE	189
11	THEATER TROUPE	364	26	BRUSSELS	185
12	SPAIN	325	27	TELEVISION CHANNEL	180
13	MUSEUM	311	28	CHOIR	176
14	BANK (FINANCIAL INST.)	299	29	TURKEY	170
15	SPORTS TEAM	295	30	CYCLE RACING	168

As is to be expected, most of these terms represent high-level concepts and can thus be used to partition the LT-space. Examples of these are *author*, *television programme*, *journal*, *river*, *sports team*, ... Nevertheless the list also contains LTs that easily lend themselves to be grouped in new higher-level concepts. This is especially true for the case of *france*, *usa*, *belgium*, *great-britain*, ... Even though the thesaurus contains a broader term relationship for these LTs to *eu-rope* and to *north-america* respectively, there is no top-level concept that also connects the latter two (e.g. *continent*, *geographic region*).

Delving into the relationships we found that 91,686 (62.27%) of the instances were dedicated to sustaining the hyponymy/hypernymy taxonomy (NT-31.07% / BT-31.20%). This was an expected result since the BT/NT couple is semantically the richest and most useful relationship available. The slight mismatch in percentages was a result of several data-entry errors (e.g. a space added to the countering LT). The synonymy/homonymy pair accounted for 49,018 (33.29%) instances (USE-16.64% / UF-16.65%) which was also an expected result. The most general relationship type available is the RT-relation. We counted 6,234 (4.23%) of the relationship instances that were of this semantically broad type. This low figure seems to indicate that very few relationship occurrences didn't fit into the other categories. As we will explain in the following section this isn't the case since it is mainly a result of several abuses of the semantics of the other relationships. To clarify the intended meaning of LTs one can make use of the SN-relationship. In this case only 307 (0.21%) of the relationship instances were allocated to this purpose. This is regrettable as scope notes are highly useful

for determining the intended sense of a lead term. We have summarized these findings in Figure 1.

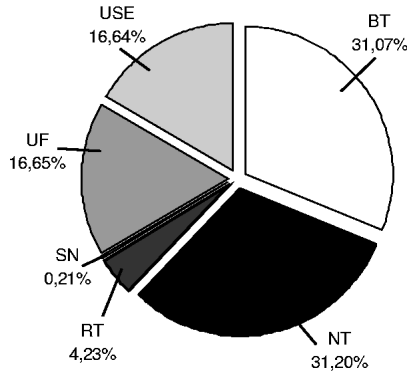


Fig. 1. Distribution of the different relationship types

2.2 Identification of Shortcomings

The shortcomings we present in this section are mainly a result of the evolution of the search-requirements since the conception of the thesaurus infrastructure. As we already mentioned, the application is under constant pressure from the programme makers to accommodate more directed and ‘knowledgeable’ queries on the archive. This has resulted in a situation where the thesaurus as a medium (not the content!) has become inadequate. Nevertheless to keep supporting the daily operation of the programme makers, it was necessary to creatively bend the rules in anticipation of a newer version. This lead to a number of conceptual lapses and inconsistencies which we will illustrate in the following subsections. For the sake of this discussion we have grouped them into three categories : lead term, taxonomy, and semantical problems.

Lead Term Problems. With regard to the lead terms we principally identified two major problems. First of all since a thesaurus follows a term-based philosophy (in contrast to a concept-based philosophy), ambiguities may arise when confronted with homonymous terms. In the thesaurus they have countered this problem by adding a context annotation between braces. An example of such an LT is *casino (film title)*, *casino (superstore)*, *casino (cycling team)*, and *casino (gambling house)*. In our case we counted 2,297 LTs that have such a suffix (approximately 1 % of the LT collection). The major problem with such suffixes is that, as reported in [5], they also can become ambiguous. What would happen if another film is made with the title *casino* for instance? This is why we have chosen to follow a concept-based approach in the new infrastructure (each concept receives a unique ID to which multiple labels can be attached).

Secondly, a problem poses itself since given names are recorded as ‘first-order’ lead terms. Unfortunately these are not grouped under a higher-level LT which makes it impossible to distinguish real LTs from these given names. For example there is no broader term connection between *deridder* and *person*³. Moreover there is a conceptual misuse of the homonymy/synonymy relationship to capture possible type errors by users of the thesaurus. This manifests itself for example in a ‘use for’ relation between the given name *d’hooge* and *d’hooghe*. This makes it impossible to distinguish the real LT from the ‘fake’ LT afterwards.

Taxonomy Problems. One of the strengths of a thesaurus is the use of ‘broader term’ and ‘narrower term’ relationships⁴ to establish a rich taxonomy. This enables the end-user to navigate through the archive by following the different layers of abstraction. A major problem we encountered was the frequent unnatural use of both relationships.

It is clear that an *investor* is a ‘broader term’ of a *stockholder*. But when we encountered *beenhouwersstraat* and *vrije universiteit brussel* amongst others as ‘narrower terms’ of *brussels* we couldn’t help but frown. This is clearly a deterioration of the knowledge since you lose the information that the first LT is a street and the second LT is a university (both located in Brussels). This is of course a result of the collection of semantically poor relationships in a thesaurus. If it were possible to connect these terms with ‘better suited’ relationships, this would enable more advanced queries on the archive (e.g. I’m looking for an archive item about a university in Belgium).

We also observed the use of BT/NT relations to indicate an ‘instance-of’ relationship. Consider for example the NT relation between *author* and *dostojevski fjodor*. This clearly isn’t a narrower term but if no other mechanism is available it is an inevitable solution. In relation to this instance-of issue we would like to raise another issue. As we will discuss in Section 4 our work will be used in a general content management system to support the broadcasters daily operation. In this case there will exist different viewpoints on the same set of meta-data. What will be experienced by one user as an instance/class, is not necessarily true for another end-user. In anticipation of the future system we have decided to follow a prototype-based approach to represent the concepts in the new infrastructure. Related problems have already been reported by [11][12].

Semantical Problems. Searching the digital archive is mainly done by composing a set of lead terms that (1) clearly defines the archived items you are interested in, and (2) reduces the size of the search result. Composing a good set

³ In some exceptional cases we did find a reference to the fact that the LT was meant to be a given name (in some cases a scope note or context suffix between braces was found). Unfortunately there was no consistent system that would allow us to automatically extract the LTs.

⁴ We define a ‘narrower term’ as a term that has a narrower meaning than the reference term. For example *mosque* is a ‘narrower term’ of *religious building*. The inverse, a ‘broader term’, is defined similarly.

of lead terms is done by browsing the thesaurus. Therefore it is crucial to be able to deduce the semantics of the chosen lead terms. ‘Scope notes’ are particularly useful for this purpose since they could for instance contain a dictionary entry that explains the intended meaning. Unfortunately as we indicated in an earlier subsection they are sparsely used. Another way to discover the meaning of an LT is to look at its relative position to other LTs (i.e. follow the relationships). The most useful for this purpose is the NT/BT couple. But as we already stated, the arbitrary use of this couple could often lead to semantical confusion. The lead term suffixes are in some cases very useful, but they are not consistently used. The entry *antwerp* (*city*) is a good illustration since most other names of cities are lacking this suffix, and hence obscure this important information. Following the ‘use’ and ‘use for’ relations is also an interesting path to follow. It relates the current lead term to synonymous terms, but as we saw earlier it is also abused for other purposes.

Generally speaking, if one wants to deduce the intended meaning of an LT one often falls back on an ad hoc combination of the above. It is only fair to say that this is mainly a result of the shortcomings of the existing thesaurus infrastructure (for this kind of application). Many of the problems we have reported are often the result of wanting to support queries that could only be answered by a detailed knowledge-base. So where is the barrier? Where does the thesaurus end and does the knowledge-base / expert-system / ... begin? It would be nice (to support these so-called ‘knowledgeable’ queries) for instance to be able to search for a multimedia item about “persons that are in their mid-thirties and that have authored at least two children’s books”. In this case we would have to be able to attach several attributes to the lead terms. And if we look into the thesaurus we sometimes find a ‘hack’ to be able to record this kind of information. It is clear that a new kind of infrastructure is needed, which we will discuss in the following section.

3 Towards an Ontology-Based Infrastructure

In this section we will briefly sketch the basic infrastructure we created as a proof of concept as well as a number of conceptual insights. The experimental prototype we created consists of an ontology-environment in Java that uses an RDBMS (PostgreSQL) to store the concepts. It contains an importer which makes it possible to convert the data dumps of the existing thesaurus. We have also included an OWL-exporter [10] for compatibility with other tools. A primitive graphical representation of the concept networks was also implemented. For the ontology we have chosen to follow a prototype based view. In practice this means that all concepts are instances. To create a new concept you can either clone an existing one or build one from scratch. It should be mentioned that we never had the intention to build a full-fledged ontology tool. We merely tried to explore the possibilities of an ontology-based infrastructure.

To get a better idea of what was needed we initially performed a number of small test-runs with existing ontology tools such as Protégé2000 [6], SoFaCB [2], ...

A major issue we were confronted with was scalability. Loading the 229,860 lead terms ('naively' promoted to concepts) immediately made clear that some of these tools were not up to the task (at that time⁵). This was not only an issue of the (in memory) data storage but also of the user interface. Take for example the standard tree-view component for showing concepts : it simply didn't scale up. A solution for this was to introduce an intelligent caching mechanism that loaded a limited range of concepts on a need-for basis. Also when zooming in on a highly connected concept (e.g. *belgium*) it became clear that visualizing (e.g. in DHTML) all connections on-the-fly is infeasible. Summarizing the number of connections (e.g. "this concept has n NTs and m RTs") and caching the visual representation on disk appeared to be a workable approach.

In our context an ontology-based application could be used in two different ways : as a *controlled vocabulary*, and as a *provider of concept templates*. The first corresponds to the use of the existing thesaurus for annotating the multimedia items in the archive. To enable this it is of vital importance to be able to upgrade the existing thesaurus content to an ontology. After all it is unfeasible to perform a manual re-indexation of the archived items. A meticulous restructuring and cleansing of the lead term collection imposes itself. A very promising approach to support the thesaurus administrators in this process is [7] [3]. However we still have to evaluate the actual use on such a huge collection. The existence of a controlled vocabulary that goes beyond the enterprize-boundaries of the broadcaster is already envisioned. This should enable the external content providers (e.g. independent production houses) to accompany their products with a set of annotations in this "unified" vocabulary. This would greatly reduce the chore of manual in-house annotation, and would certainly improve the quality of the meta-descriptions. In order to succeed in setting up such an ontological commitment between the different media partners, it is crucial to create a good set of upper ontologies (cf [4]). As we already mentioned in an earlier section, we believe that an in-depth analysis of the existing thesaurus could identify possible candidate concepts for this level. For this purpose we are currently including a number of analysis functionalities in the experimental environment (e.g. on-the-fly generation of Table 2.1, extraction of LT suffixes, ...). This should help the thesaurus administrators to make sound choices to compose the set of core concepts.

The second use of an ontology as a concept-template provider is a separate issue. To accommodate the richer set of archive queries one also has to provide a richer set of knowledge in the annotations. For this purpose it is necessary to be able to 'instantiate' concepts in the ontology. In our approach this is done by cloning existing prototypical concepts (i.e. these form the ontology for the 'instantiated' concepts) . An example of such a concept could be *person* in which we foresee slots for *name*, *date-of-birth*, ... This is very similar to the traditional

⁵ Since the start of this project, a number of ontology tools have been greatly improved. Based on our own recent findings and the evaluation results reported in [1] we currently consider using the KAON tool suite [8] for future experiments.

class/instance creation in knowledge-bases. It is clear that this will certainly improve the support for advanced queries. However it remains unclear how far the broadcaster should go in setting up such a broad encyclopedic knowledge-base. Also in this case the need for a shared ontology between the different media partners arises. The knowledge-base would consequently contain pointers to the relevant ‘out-house’ knowledge bases.

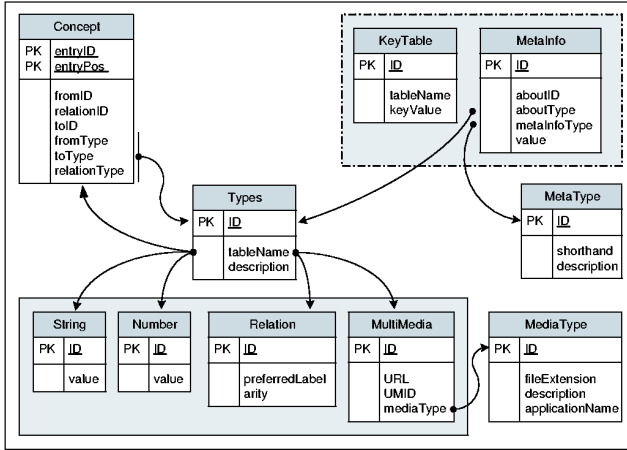


Fig. 2. Conceptual schema of database

The database we use to store the concepts in the ontology is based on the schema presented in Figure 2. Central in this schema is the table *Concept* which is used to establish the concept network. A concept could be related to another concept (e.g. a BT relationship) or to a terminal type such as *String*, *Number*, ... To distinguish between these we have included a foreign key (*fromType*, *toType*, *relationType*) that points to the corresponding type-table. This makes it possible to support new types without changing the existing conceptnetwork. An example conceptnetwork is shown in Figure 3. The *MetaInfo* table is used to record all kinds of meta-information such as the date an entry was created/updated/. . . . We have found that this scheme is quite robust to change since it is very generic. The downside to this is that a lot of the semantic behavior has to be captured outside the database (such as the interpretation of a BT/NT relationship).

As we already stated we initially converted the thesaurus in a naive way. This resulted in three *Concept* table entries per LT, which boils down to approximately 700,000 tuples. Adding new relationships between LTs will surely augment this number.

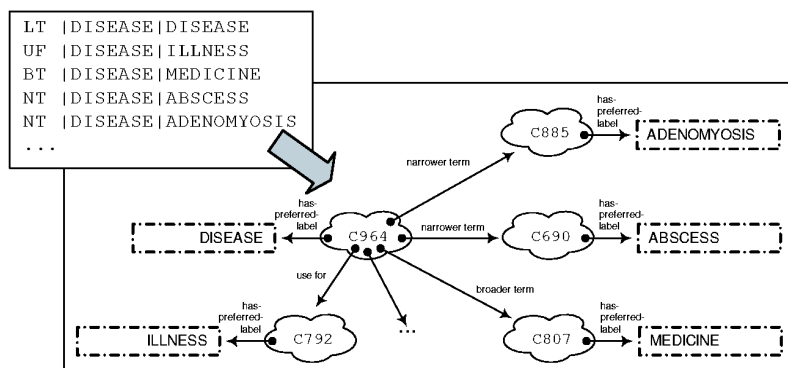


Fig. 3. An example conceptnetwork for *disease*. A cloud represents a concept, the arrows indicate the relationships

4 Future Work

Search Heuristics. We have already experimented with the use of search heuristics on the ontology. In the particular experiment we kept track of the frequency that certain lead terms were used. This was consequently used to compose ‘virtual’ groupings of terms in the ontology. These virtual groupings were initially implemented in an extensional way by enumerating the different LTs that belong to it. This kind of grouping is mainly interesting to record static information (e.g. a thematic grouping). In the future we want to explore intentional groupings which provide a way to dynamically group LTs.

Temporal Information. Currently the thesaurus doesn’t contain references to temporal information. The main reason for this is that there was no ‘clean’ way to include it in the existing infrastructure. It is however seen as highly relevant for the archive queries. An example of this is information about the period when a certain person was president of a country. The availability of a mechanism to store temporal information would also make it possible to get rid of situations with unclear LTs. We find for instance the LTs *germany1* and *germany2* in the existing thesaurus to indicate germany before and after the unification of east and west. As a consequence all links that are relevant in both situations are recorded twice (in some cases this isn’t the case and hence entering the search path from one or the other could lead to different results!).

Enterprise-wide Content Management System. As we mentioned in the introduction, the broader context of this work is the installation of a general content management system to support the activities of a radio and television broadcaster. This is strongly related with Enterprise Application Integration. The major difference is that a lot of work involved in the production of radio

and television has a creative (artistic) nature. Consequently each team has its own approach and set of tools. It is clear that it is out of the question to enforce a unified view / tool suite on these production activities (this would restrain their creativity!). Moreover such a de facto central alignment is unmanageable in this context as a result of the scale and scope of this business's nature. Nevertheless from a management perspective it is crucial to have an integrated and controllable system of data and processes. We believe we can find a compromise in the semantic web vision to meet these contradicting requirements. Instead of a tightly coupled system (e.g. ERP systems) this would enable setting up a loosely coupled 'intranet-alike' system (cf the internet). For our work package this would mean that a much richer set of meta-data could already be captured at production-time instead of at archival-time (e.g. GPS data indicating the recording location, lenses used, participants, scenario, ...).

5 Conclusion

In this paper we have reported on a number of experiences when confronted with the revaluation of a large-scale thesaurus. The revaluation of this thesaurus was mainly driven by an urgent need to support more advanced and 'knowledgeable' queries on a vast multimedia archive in the context of a radio and television broadcaster.

During our analysis of the existing infrastructure we found that there were several conceptual lapses and inconsistencies in this thesaurus. These were mainly the result of its inadequacy as a medium to support the daily operation of the programme makers. The problems we identified were related to lead terms (suffixes to resolve ambiguity, given names as first-order lead terms), to the taxonomy (misuse of broader term / narrower term), and to the semantics of the lead terms (intended meaning of a lead term). This identification has resulted in a number of guidelines which will be used to support the restructuring work of the thesaurus administrators.

To experiment with the possibilities of a next generation infrastructure we have approached the thesaurus from an ontology perspective. For this purpose we have built an experimental environment into which we converted the thesaurus to a prototype based concept network. Even though preliminary, this has resulted in insights with respect to scalability issues (in memory storage, user interface) and more conceptual themes. With respect to the latter, it became clear that in our context, an ontology-based application could be used as a controlled vocabulary as well as a provider of concept templates. Moreover exploring the use of the conceptnetwork schema has lead to a deeper understanding of issues related to genericity and meta-circularity (e.g. defining the semantics of the relationships inside the database itself).

Specifically in the context of an enterprize-wide content management system these insights will certainly influence our future directions.

References

1. Angele, J. and Sure, Y. (eds): EON 2002 - Evaluation of Ontology-based Tools. In: EKAW 2002 Workshop Proceedings (2002)
2. Deridder, D. : A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. In: Proceedings of the 5th Joint Conference on Knowledge-based Software Engineering. IOS Press (2002)
3. Gangemi, A. and Guarino, N. and Oltramari, A. and Borgo, S.: Cleaning-up WordNet's top-level. In: Proc. of the 1st International WordNet Conference (2002)
4. Gangemi, A. and Guarino, N. and Masolo, C. and Oltramari, A. and Schneider, L.: Sweetening Ontologies with DOLCE. In: Proceedings of the International Conference on Knowledge Engineering and Knowledge Management. AAAI (2002)
5. Grefenstette, G.: Explorations in Automatic Thesaurus Discovery. Kluwer Academic Publishers (1994)
6. Grosso, W.E. and Eriksson, H. and Ferguson, R.W. and Gennari J.H. and Tu, S.W. and Musen, M.A. :Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000). SMI Report SMI-1999-0801 (1999)
7. Guarino, N. and Welty, C. A. : Evaluating ontological decisions with OntoClean. In: Communications of the ACM 45-2(61-65) (2002)
8. KAON – The Karlsruhe Ontology and Semantic Web Framework. FZI (Research Center for Information Technologies) and AIFB (Institute of Applied Informatics and Formal Description Methods) - University of Karlsruhe.
<http://kaon.semanticweb.org/>
9. Maedche, A. and Motik, B. and Stojanovic, L. and Studer, R. and Volz R.:Ontologies for Enterprise Knowledge Management. In:IEEE Intelligent Systems. 1094-7167/03. IEEE Computer Society (2003)
10. Patel-Schneider, P. F. and Hayes, P. and Horrocks, I. : OWL Web Ontology Language Semantics and Abstract Syntax. W3C Working Draft 31 March 2003.
<http://www.w3.org/TR/owl-semantics/>
11. Welty, C.A.: Towards an Epistemology for Software Representations. In: Proceedings of the 10th Knowledge-Based Software Engineering Conference. IEEE Computer Society Press (1995)
12. Welty, C.A. and Ferruci, D.A.: Classes in Software Engineering. In: Intelligence. Summer (1999) 24-28

On Storing and Querying XML Data within Network-Enabled Embedded Systems

Hyoseop Shin

Software Center
Samsung Electronics Co., LTD.
Seoul, Korea
hyoseop@samsung.com

Abstract. Modern digital consumer devices are now enabled to handle certain types of XML data. In many cases, such data are periodically downloaded from remote via some network. In this paper, methods of storing and querying of XML data within digital devices are considered where relatively large scale XML data are downloaded in packetized formats for secure and efficient delivery from the senders to the receiving devices. Specifically, the downloaded XML fragments are natively organized by the fragment-oriented data structures and the supporting indexes both of which can be easily implemented on top of a lightweight relational storage. The proposed design provides efficient search and retrieval of XML data as well as dynamic update and versioning within resource-limited digital devices.

1 Introduction

XML [2] is increasingly being exploited not only in large scale enterprise systems but also in compact scale embedded systems. Modern digital consumer devices such as set-top boxes, mobile phones, PDAs, and home network servers have now come to handle certain types of XML data. Within consumer devices, XML can be used for exchanging arbitrary personal data such as phone number list, or for describing and managing multimedia (e.g., mp3, image, video, TV programs) that the device can handle. In many cases, such data are downloaded from remote to the digital devices for use via some network. Some data are transformed into an internal format, but some are stored in native XML format for further processing such as searching, browsing and updating of data.

In this paper, methods of storing and querying of XML data within digital devices are considered where relatively a large amount of XML data are downloaded in packetized formats for secure and efficient delivery from the senders to the receiving devices. Specifically, the downloaded XML data fragments are organized by the fragment-oriented data structures and the supporting indexes both of which can be easily implemented on top of a lightweight relational storage.

The remainder of the paper is organized as follows. Section 2 describes our problem definition. Section 3 summarizes the related work. Section 4 presents the fragmentation scheme of XML data. Section 5 gives the indexing scheme of

XML data. Updating and retrieval of XML data are explained in section 6. We give our conclusion in section 7.

2 Problem Definition

In this section, we introduce an application in digital television broadcast environments where XML is used as a metadata language to describe TV contents as a motivating example.

In digital TV broadcast environment, not only audiovisual contents but also descriptive data about contents are transmitted via a broadcast medium to support electronic searching and browsing of TV programs and other EPG(electronic program guide) scenarios within set-top boxes. As the de-facto standard in describing these descriptive data (a.k.a *metadata*) about audiovisual contents, XML is being increasingly used(e.g. TV-Anytime Metadata [8] and MPEG-7 Multimedia Description Schemes [1]). The metadata regarding TV programs can include various kinds of information such as information about programs, program groups, program reviews, credits, program locations, and service stations. Below is an example of TV-Anytime metadata that consists of a program information, a broadcast event, and a service information.

```
<TVAMain>

<ProgramInformationTable>
  <ProgramInformation programId="PROG-123">
    <BasicDescription>
      <Title>Sunrise News</Title>
      <Synopsis>Morning News</Synopsis>
      <Keywords>
        <Keyword>politics</Keyword>
        <Keyword>economy</Keyword>
      </Keywords>
      <Genre>News</Genre>
      <CreditsList>
        <CreditsItem>
          <Role>Reporter</Role>
          <Agent>Richard Perry</Agent>
        </CreditsItem>
        <CreditsItem>
          <Role>Producer</Role>
          <Agent>Tom Martin</Agent>
        </CreditsItem>
      </CreditsList>
    </BasicDescription>
  </ProgramInformation>
</ProgramInformationTable>
```

```

<ProgramLocationTable>
  <BroadcastEvent serviceIdRef="NBC">
    <ProgramIdRef>PROG-123</ProgramIdRef>
    <PublishedTime>2003-04-29T09:40:00</PublishedTime>
    <PublishedDuration>PT30M0S</PublishedDuration>
  </BroadcastEvent>
</ProgramLocationTable>

<ServiceInformationTable>
  </ServiceInformation serviceId="NBC">
    <Name>NBC</Name>
    <ServiceURL>www.nbc.com</ServiceURL>
  </ServiceInformation>
</ServiceInformationTable>

</TVAMain>

```

As the XML metadata document transmitted by the broadcaster is composed of various parts and may contain considerable amount of information about the TV programs of a certain interval (e.g., two weeks), it is split into smaller pieces of *fragments* to be transmitted over a broadcast network with a bandwidth limit. For the example above, the *ProgramInformation*, the *BroadcastEvent*, and the *ServiceInformation* are separated into different fragments for delivery.

The advantages that the entire XML metadata document is fragmented into smaller parts when it is to be transmitted are summarized into two. First, the document can be more securely transported over error-prone networks in many small fragmented packets rather than in one large packet. Second, updates can be done on the portions of the document. When the entire document is transmitted at once, the whole document should be re-transmitted whenever a portion of the document changes. However, in fragment-based delivery, only the changed fragments have to be re-transmitted. In our example, the fragments are transmitted repeatedly over a carousel network so that a set-top box can receive the necessary TV information at any moment. The fragments of the currently airing program are sent more frequently than those of the future programs. Updated versions of the same fragments can be sent due to the change of TV program schedules. Fig. 1 illustrates the transmission process of fragmented and encapsulated XML metadata.

As our example implies, in order to efficiently and securely deliver large XML documents over bandwidth-limited and error-prone networks, it is crucial to split the documents into smaller pieces and to transmit the documents in the fragment units, not in the entire document units.

Meanwhile, this fragment-based delivery of XML documents also requires receiving devices to handle the documents in a compliant way. Specifically, the local XML storage system in a receiving device is required to support:

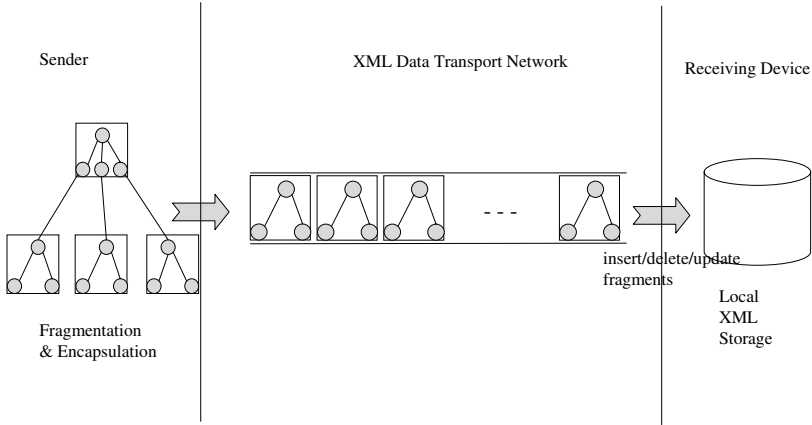


Fig. 1. Transmission of Fragmented XML Metadata

- Efficient reconstruction of the original XML documents
- Dynamic updating and versioning of portions of the XML documents
- Efficient search and retrieval of portions of the XML documents

To meet these requirements, we propose a fragment-oriented XML storage architecture for embedded systems.

3 Related Work

Many methods of storing and querying XML data have been proposed in the literature [14,11,13,3,9,10,12,5,4,7,6]. As our fragment-oriented scheme can be established mainly on top of a lightweight relational storage, we summarize well-known methods of storing and retrieving XML data in the relational databases and discuss their advantages and disadvantages.

3.1 Node Numbering Schemes

Several methods have been reported on storing and querying XML data in relational databases. Among those methods, variations of the node numbering scheme [14,11,9] are most popular as it is quite natural to store each element (and attribute) in XML documents as a relational tuple.

In the node numbering scheme [14], each element or attribute in the XML document is separately stored as a tuple within a relational table. Each tuple has the structure of $\langle doc\_id, begin\_pos, end\_pos, level \rangle$, where *doc_id* is a document identifier, and *begin_pos* and *end_pos* are the beginning and ending word positions of a tag within the document, and *level* is the depth of a tag when the XML document is represented as a tree.

Consider the *ProgramInformation* of the XML example in section 2. It can be represented as a tree shown in Fig. 2 according to the node numbering scheme.

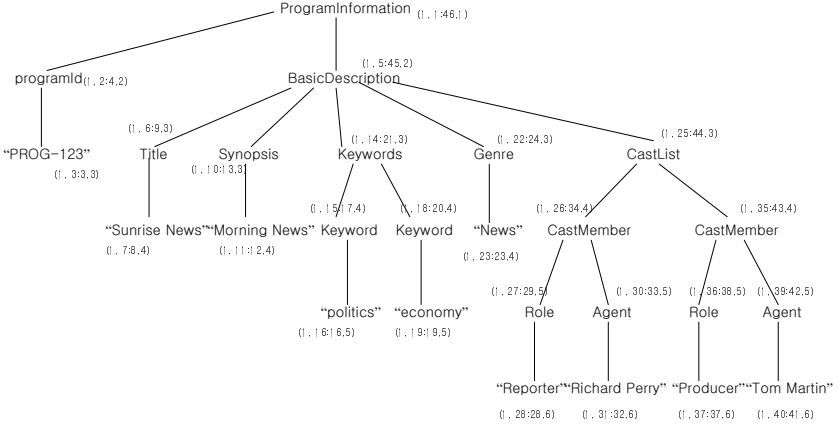


Fig. 2. Tree Representation of an XML document by a Node Numbering Scheme

In this figure, nodes of the document were labelled by visiting in pre-order. The *doc_id* of each node is 1 because all of them are contained in the same document. Leaf nodes are treated as a special node type that stores the text value of the parent node.

Under the numbering scheme, the ancestor-descendant and parent-child relationships between nodes, which are represented as the axes '/' and '//', respectively, in XPath or XQuery, can be verified by examining the tuple values. If a node 'A' is an ancestor of a node 'B' (i.e., A//B), the conditions ' $A.doc\_id == B.doc\_id$, $A.begin\_pos \leq B.begin\_pos$, $A.end\_pos \geq B.end\_pos$ ' are satisfied. If a node 'A' is the parent of a node 'B' (i.e., A/B), the condition ' $A.level == B.level - 1$ ' is added to the conditions above. According to these properties, to obtain node pairs that satisfy the 'A//B' or 'A/B' relationships, a structural join operation on the node table is performed using the join conditions above.

The node numbering scheme has the advantage that it can be used for storing generic XML data even without a DTD or XML schema for the input data. However, this method suffers from several disadvantages in querying and updating XML data within embedded systems. In processing queries, the node numbering scheme causes as many consecutive joins as the length of a path expression which is an essential component in XML queries. The performance degradation can be especially severe in embedded systems where the join operations get slower due to the frequent I/Os of hard disk caused by the shortage of main memory. Reconstruction of XML data from a database is also costly because every element is mapped into a tuple and thus raises several joins to re-assemble them. Update

operation is not efficient either, because an update to a node can trigger updates to its parent nodes.

3.2 Attribute Inlining Scheme

The attribute inlining scheme [12] may overcome excessive fragmentation of XML documents by assembling multiple XML elements into attributes of a same relational table. The method *inlines* as many descendants of an element as possible into a single relation unless a descendant node has an in-degree greater than one or zero. Otherwise, plural descendants will create another relations.

For the *ProgramInformation* of the XML example in section 2, the resulting database will consist of the following relational tables by the attribute inlining scheme:

```

ProgramInformation (id, programid, basicdescription.title,
                  basicdescription.synopsis)
Keyword (id, parentid, keyword)
Genre (id, parentid, genre)
CastMember (id, parentid, castmember.role, castmember.agent)

```

In this example, *BasicDescription.Title*, *BasicDescription.Synopsis* have been inlined into the table *ProgramInformation*, but *Keyword*, *Genre* and *CastMember* have been detached from *ProgramInformation* into another tables because they can be plural (e.g. there can be more than one keywords for a TV program.). Internal nodes like *Keywords* and *CastList* are not explicitly represented.

The attribute inlining scheme can automatically generate a relational database schema for XML documents based on a given DTD or XML Schema. Also, it may suffer less from several joins in processing path expression queries than the node numbering scheme. In practice, however, the resulting schema may not be considerably different from that of the node numbering scheme, especially when many of nodes in a DTD specification have in-degree greater than one. In our application, it turned out that the TV-Anytime metadata [8] contains about 80% plural XML tags. Another serious problem with the attribute inlining scheme is that the approach is so sensitive to the XML data schema that a change of a node would trigger the rebuilding of the database.

To overcome these problems found in the previous work, we propose a fragment-oriented XML storage scheme for domain-specific embedded systems.

4 Fragmentation Scheme

In order to conform to the transmission of XML data in the fragment basis, our XML repository stores the XML data in the same fragmented form as it is transmitted. The policy how the XML data is fragmented can be pre-defined in the domain as our motivating example or can be determined arbitrary by the repository unless there is a pre-defined fragmentation policy.

The hierarchical structure of the XML data in a domain is divided into a set of fragment types. A fragment type can be located under (or a child of) another fragment type. In the example of section 2, there are four fragment types: *TVAMain*, *ProgramInformation*, *BroadcastEvent*, and *ServiceInformation*. An actual XML instance is split into fragments each of which belongs to a fragment type in accordance with the determined fragmentation policy. As fragments should be independently updated, a parent/child relationship between two nodes is not maintained unless they belong to the same fragment type. Thus, in order to maintain the parent/child relationship between specific two nodes in XML data structure, the fragmentation should be designed so that they are located in the same fragment type or an *IDREF* node should be explicitly added in the XML document to refer to one node from the other node.

In Fig. 3, the parent/child relationship between the node *a* and *b* will be maintained because they remain in the same fragment type, while the relationship between the node *b* and *c* are no longer maintained after fragmentation. Those links that connect the nodes which are located in different fragment types are defined as *weak links*. A fragmentation policy can be based on finding weak links in the XML documents.

The fragments are stored in a relational table. The *fragment* table contains *fragment_id*, *fragment_type*, *fragment_version*, *fragment_content* fields. The *fragment_id* is a fragment identifier which is unique within the transmission layer and the local repository. The *fragment_type* indicates to which fragment type a fragment belongs. The *fragment_version* is the current version number

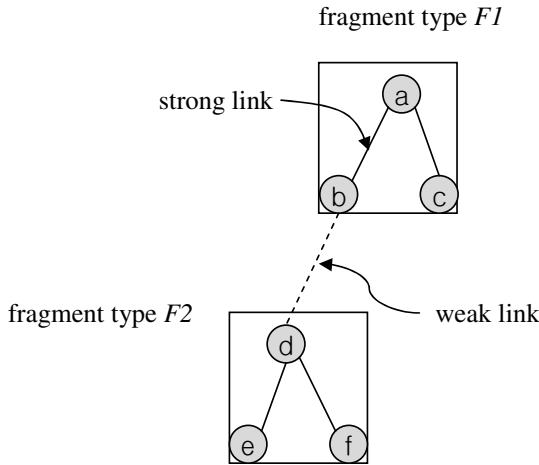


Fig. 3. A Weak Link Between Fragment Types

of the fragment. The *fragment_content* is the content of a fragment itself which may or may not be a parsed form.

5 Indexing Scheme

Meanwhile, it is not efficient to look into the content of every fragment stored in the *Fragment* set only to find some fragments that match conditions specified by values of certain nodes of a fragment type. To avoid this overhead, frequently referred nodes in query conditions are indexed in advance. Indices are defined on a fragment type using single or multiple nodes within a fragment type as the key. For the XML example in section 2, an index can be defined on *ProgramInformation* fragments using the element *Genre* to accelerate searching the information of the programs of specific genres.

An index consists of a set of entries each of which is formatted as $\langle key\_value, fragment\_id \rangle$. The *key_value* represents the value of the index key. If an index key is a composite type, the *key_value* consists of multiple fields. The *fragment_id* is the identifier of a fragment. An index of XML fragments can be built in common access methods like B-trees. As there can be more than one index in the repository, two supplemental structures are provided to assist selecting indices for user-given queries: *IndexDictionary*, *IndexKeyfield*. The *IndexDictionary* maintains the information of each index available in the repository. The structure of the *IndexDictionary* is $\langle index\_id, fragment\_type, num\_key\_field, key\_field\_list, index\_version \rangle$. The *index_id* identifies an index. The *fragment_type* represents the type of fragments on which an index is built on. The *num_key_field* is the number of key fields which constitute the index key and the *key_field_list* lists the key fields of an index. The *index_version* represents the version number of an index. Versioning of indices is necessary if XML fragments as well as their indexing data are transmitted from the sender to the receiver. In case the receiver cannot be capable of building and updating indices for itself due to the lack of computing resources, the sender may transmit necessary indices to the receiver and the receiver updates the indices, if necessary, using the version information.

The *IndexKeyfield* describes the information of the key fields used in the index keys. The structure is $\langle field\_id, field\_path, field\_type \rangle$. The *field_id* is the identifier of a key field. The *field_path* represents the absolute path from the root node to the designated node. The *field_type* is the type of the node in the repository. Fig. 4 illustrates the index and fragment structures of the repository.

6 Update and Retrieval

6.1 Updating XML Data

The repository supports updating XML data in the *fragment* basis. Versioning is applied when updating a fragment. Once a fragment is about to be updated, the associated indices are also updated accordingly.

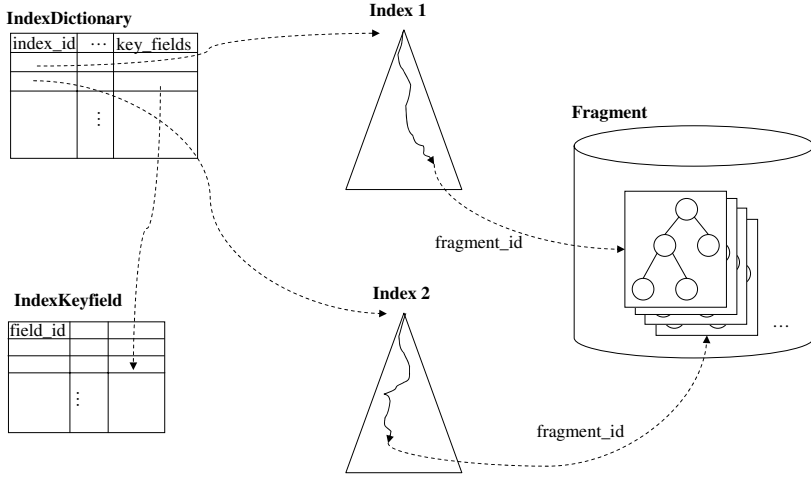


Fig. 4. Index and Fragment Structures

When a new fragment arrives at the repository to be updated, the identifier, type, and version of the fragment is identified. If a fragment of the same identifier exists in the *Fragment* set, the *modify* process begins. Otherwise the *insert* process begins.

In the *modify* process, the versions of the existing fragment and the newly coming one are compared together, and if the latter is a new version, the former is replaced with the latter. Otherwise, the latter is just discarded. In the *insert* process, the newly coming fragment is inserted into the set *Fragment*.

When updating a fragment, the indices associated with it are also updated. An index is associated with a fragment if the fragment type for which the index is built is the same as the one the fragment belongs to. When a fragment is inserted, an entry $\langle key\_value, fragment\_id \rangle$ is inserted into the associated index, where the *fragment_id* is the identifier of the fragment and the *key_value* is the value of the key field of the index for the fragment.

When a fragment is modified, the index entry of the fragment is removed first from the associated indices, then insertions are made to the indices. Note that if a fragment is to be updated, it is necessary for the fragment to get parsed because *key_value* should be identified for updating the associated indices.

When a fragment is deleted, index entries of its associated indices are also deleted.

6.2 Retrieving XML Data

Like update operations, the XML data stored in the repository are retrieved in the fragment basis. A fragment type is specified as the query target and query conditions are specified against the values of the nodes within the fragment type.

For example, the fragments in the fragment type '*ProgramInformation*' can be filtered on their genres. After the matching fragments are fetched from the *Fragment* set, they get parsed into a form as desired in the applications. The retrieval process consists of four steps:

- 1) **Selecting Indices** select the suitable indices to process the user-given query by looking into the *IndexDictionary* and *IndexKeyfield* structures.
- 2) **Filtering Fragment Identifiers** filter the identifiers of the fragments that match the query condition by looking into the relevant indices.
- 3) **Retrieving Fragments** retrieve the content of the fragments indicated by the fragment identifiers by looking into the set, *Fragment*.
- 4) **Parsing Fragments** parse the content of the fragments which are returned as the query result.

7 Conclusion

This paper presented an XML repository for embedded systems. The repository can support dynamic update, versioning, and efficient retrieval of XML data in the fragment level. Considering carriage of XML data over bandwidth-limited and error-prone networks, the XML data are supposed to be in a fragmented format for secure and efficient delivery. The proposed repository conforms to this fragmentation scheme for XML data, and thus is expected to provide better functionality and performance over generic methods of storing and querying XML data in network-enabled environments. The repository has been designed so that it can reside in resource-limited embedded systems and can be easily implemented on top of a lightweight relational database engine. Future work includes application of the proposed system to various domains and intensive performance evaluations under various environments.

References

1. ISO/IEC 15938-5. *Information Technology – Multimedia content description interface – Part 5: Multimedia Description Schemes*. 2001.
2. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation*. World Wide Web Consortium, October 2000.
3. Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras, and Carlo Zaniolo. Efficient structural joins on indexed XML documents. In *Proceedings of the 27th VLDB Conference*, pages 263–274, Hong Kong, China, August 2002.
4. Brian F. Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *Proceedings of the 26th VLDB Conference*, pages 341–350, Roma, Italy, September 2001.
5. Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing semistructured data with stored. In *Proceedings of the 1999 ACM-SIGMOD Conference*, pages 431–442, Philadelphia, USA, June 1999.

6. Mary Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the 1998 IEEE International Conference on Data Engineering*, pages 4–13, Orlando, Florida, February 1998.
7. Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd VLDB Conference*, pages 436–445, Athens, Greece, August 1997.
8. TV-Anytime Metadata Working Group. *Specification Series:S-3 on Metadata Version 1.3*. TV-Anytime Forum, January 2003.
9. Torsten Grust. Accelerating XPath location steps. In *Proceedings of the 2002 ACM-SIGMOD Conference*, pages 109–120, Madison, Wisconsin, June 2002.
10. Torsten Grust, Maurice van Keulen, and Jens Teubner. Staircase join: Teach a relational DBMS to watch its (axis) steps. In *Proceedings of the 28th VLDB Conference*, Berlin, Germany, September 2003.
11. Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In *Proceedings of the 26th VLDB Conference*, Rome, Italy, September 2001.
12. Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of the 25th VLDB Conference*, pages 302–314, Edinburgh, Scotland, September 1999.
13. Divesh Srivastava, Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, and Yuqing Wu. Structural joins: A primitive for efficient XML query pattern matching. In *Proceedings of the 2002 IEEE International Conference on Data Engineering*, San Jose, USA, February 2002.
14. Chun Zhang, Jeffrey F. Naughton, Qiong Luo David J. DeWitt, and Guy M. Lohman. On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM-SIGMOD Conference*, Santa Barbara, CA, USA, May 2001.

An Object-Oriented Invocation Layer for the Java Message Service

Klaus Jank and Roy Oberhauser

Siemens AG, CT SE 2, Otto-Hahn-Ring 6,
81730 Munich, Germany
{klaus.jank, roy.oberhauser}@siemens.com

Abstract. New applications and software environments are increasingly distributed across a large or even unpredictable number of networked computing devices, require mobile and ad-hoc networking capabilities, and must integrate with more systems, all of which create greater demands on the middleware used to realize these systems. On the Java platform, RMI is a well-established paradigm, yet deficiencies become evident in particular with regard to scalability and remote invocation completability - which is the assurance that invocations are executed according to client and service expectations regardless of the state of the participants or the communication network. While the Java Message Service (JMS) addresses these deficiencies, it lacks the simplicity, explicit contracts, clear coupling, and strong typing of an object-oriented invocation paradigm. This paper will describe our Java Invocation Layer for Messaging (JILM), a(n) (a)synchronous invocation layer to support object-oriented invocations while leveraging the unique distribution and QoS properties that JMS provides.

1 Introduction

Distributed Java computing applications have often relied on RMI or RMI-IIOP [21] (referred to as RMI in this paper) because of its object-oriented paradigm, simplicity, and wide availability. However, changing environments and demands, such as a larger and often unpredictable number of networked computing devices (e.g., internet, embedded, and pervasive computing), greater mobility and ad-hoc networking (e.g., P2P, wireless), and increasing inter-system integration (e.g., intranets, B2B) cause RMI's deficiencies to become apparent, in particular scalability and remote invocation completability.

Scalability challenges occur when addressing an unpredictable number of participants or long duration invocations due to RMI's synchronous invocation model. The advantages of asynchronicity for scalability have been investigated, e.g. for CORBA in [1]. However, there are situations where it is desirable to be able to choose the appropriate mechanism in an invocation: asynchronous invocations - to fulfill scalability requirements, or synchronous invocations - to address programming complexity or to perform short duration invocations. Thus support for both invocation models is desirable.

Moreover, if a client makes an asynchronous invocation, this should not require the service to have to support the additional complexity of asynchronicity. But as system integration increases, more services rely on and utilize other services (i.e. service chaining), where the asynchronous invocation model may be preferable¹ for the service, such as has been argued for middle-tier servers in Eckerson [5] and Deshpande [4]. Consequently, the client and service invocation models should be decoupled, supporting independent client and service usage.

With regard to remote invocation completability in our scenarios, the following invocation properties, missing in RMI, become important: Time-Independent Invocations (TIIs), Location-Independent Invocations (LIIs), group invocations, batch invocations, interim or partial results, and Quality-of-Service (QoS). These will be described below.

Since the simultaneous availability of invocation participants cannot always be guaranteed, TIIs are desirable to allow the separate parts of the invocations to be stored and forwarded when each participant becomes independently available. This decouples the client and server lifetimes. The CORBA Specification [16] includes support for TIIs.

RMI invocations rely on object references which can change in dynamic systems, causing destination-specific, point-to-point invocations to fail. Similarly, the issue with Inter-Operable References (IORs) is known for CORBA invocations [6], being addressed with Inter-Operable Group References (IOGRs) in Fault Tolerant CORBA [16]. LIIs support the completability of invocations to other available and compatible invocation targets.

Group invocations, as used in this paper, refer to the ability to have a set of services receive the same invocation. Whereas group messaging has been used to distribute events, they often lack the desired distributed-object abstraction, cp. JavaGroups [7]. And while the concept of group invocations is common in parallel programming, e.g. Group Method Invocation [12], our motivation is not parallelism for performance *per se*, but rather the assurance that the entire group eventually receives the invocation, e.g. to change the configuration parameters or distribute data. Group invocations enhance completability while supporting the simplicity of object-orientation, e.g. in unreliable networking environments and in cases when the client cannot know which objects must receive the invocation.

Batch invocations support the grouping of multiple related invocations. With regard to completability, the entire group of related requests is viewed as a single entity, thus providing consistency for the invocation participants that either all or none are sent. This is advantageous, for example, in TII scenarios when connectivity cannot be guaranteed.

Partial or interim results may be desirable in such systems when long-duration invocations or large transfers are involved. For example, interim updates to the status of a request could be provided at critical transition points (e.g., “request being processed,” “request completed,” etc.). Partial results could include large amounts of chunked detector or measurement values. Partial results enhance completability (with

¹ In order to improve scalability, concurrency or asynchronous models can be used. However, concurrency models often lead to increased overhead such as thread management, context switching, dynamic memory allocation, and synchronization [20]. Asynchronous models - where the executing thread runs on a different CPU than the invocation thread, are preferable in scenarios where blocking may occur, such as service chaining.

regard to expected behavior) by supporting “best effort,” while interim results provide the ability for clients to know what is occurring with their longer-duration request.

QoS provides the capability of specifying different quality levels for invocation delivery to support deterministic behavior, as in durability (e.g., surviving reboots), ordering (e.g., commit after set), prioritization (e.g., abort might have higher priority), reliability (e.g., retries) and deterministic semantics (e.g., exactly once). JMS [22] and CORBA AMI support similar types of properties.

While the use of messaging such as JMS instead of RMI would support the desired asynchronicity for scalability and remote communication completability, the use of messages increases programming complexity due to its lower abstraction level. The lack of an object-oriented invocation model results in implicit contracts, implicit coupling, and weak typing. This can make JMS-based systems more difficult to program and maintain correctly.

Hence, neither RMI nor JMS alone satisfies the desired properties in our context. However, the choice of a communication provider for an application in the scenarios we discussed is often critical and involves many factors. In order to further adoptability, the solution should leverage middleware capabilities already commonly available and support provider exchangeability.

Thus there exists a need for an invocation middleware that provides the simplicity and strong typing of an object-oriented paradigm while supporting the following properties:

- scalability of asynchronous invocations,
- Time-Independent Invocations (TIIs),
- Location-Independent Invocations (LIIs),
- group invocations,
- batch invocations,
- interim or partial results,
- QoS,
- client-service invocation model decoupling,
- communication provider exchangeability.

We have designed JILM as a solution to provide these properties. We will begin with a description of our general solution approach followed by a detailed solution. We will then evaluate JILM and compare it with other middleware and related work.

2 General Solution Approach

Our solution approach consists of the following participants (see Fig.1).

JMS. JMS is used to transport the call, providing asynchronous communication and making the desired QoS properties available to services, including priority, filtering, call expiration time, call persistence, durability, etc. To support TII, the calls are queued by JMS until the intended peer becomes available to receive and process the calls. Similarly, queuing supports LIIs since any message consumer may process the call and the client is unaware of the service’s true location or reference. Group

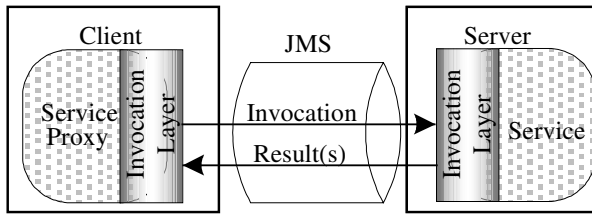


Fig. 1. General solution approach

invocations are supported by placing the calls in a topic (publish-subscribe), where multiple services retrieve the identical call. Batch invocations are supported by combining messages into a transacted session.

Invocation Layer. The object-oriented invocation layer addresses JMS deficiencies and supports decoupled client and server invocation models.

On the client, the method invocation is translated into a message that includes the service identifier, the method name, and parameters. On the server, the invocation layer retrieves the message and invokes the call on the appropriate service instance.

The following client invocation models are supported:

- *Synchronous.* Blocks the client thread until the response is received,
- *Asynchronous.* After the call is placed in a message, control is returned to the client thread while the response is received in a separate thread,
- *Asynchronous with multiple responses.* Multiple responses for the same invocation are returned in a separate thread to support interim or partial results,
- *Futures.* Creates a non-blocking call whereby the same client thread context is retained to retrieve the response. Via polling, the client can check if results are ready or block until ready.

The supported client invocation models are declared in the service proxy interfaces, which include the supported synchronous and or asynchronous method declarations. To be able to clearly distinguish a synchronous from an asynchronous invocation, separate interfaces can be used, and methods that provide an asynchronous invocation model must define a response handler as the first input parameter. Asynchronous responses are provided in the client when the invocation layer invokes the response handler.

The following server-side service invocation models are supported:

- *Synchronous.* Simplifies service programming and provides better performance for short duration calls, since it retains the thread context,
- *Asynchronous.* Enables pooling and reuse of threads that would otherwise wait, which can enhance server scalability in certain scenarios where service chaining or network calls are involved,
- *Asynchronous with multiple responses.* Multiple responses, such as partial results, may be returned before the final response.

The supported service invocation model(s) are defined by the service implementation, not the proxy. Thus a service may implement only a synchronous method, but may offer the client both synchronous and asynchronous method declarations. This decoupling of client and service invocation models can simplify the service implementation while providing flexibility for the client. Reflection, dynamic method signature mapping, and a preference strategy are used to decide which service implementation method corresponds to a client call.

Partial results are supported by associating multiple response messages with a single call context.

Service. The service provides functionality independent of the middleware used to invoke it. The service implementation defines the supported service invocation model for each method. During the registration of a service, the invocation layer uses reflection to dynamically determine the implemented method signatures in order to use the appropriate invocation model to invoke a method.

Proxy. The Proxy pattern [3] is used to represent the service on the clients. Since the mechanism for propagating a call is independent of the actual interface or method name, a `java.lang.reflect.DynamicProxy` is utilized, which supports different service interfaces with a common service proxy implementation of the client invocation layer described above. For each service, synchronous and/or asynchronous interfaces are provided (or a variant of one interface that provides both synchronous and asynchronous methods), allowing the client to choose the invocation model it wishes to use (per interface or per method).

Since the service configures the JMS-related QoS properties of the service proxy instance, by distributing the proxy, e.g. via Jini's Lookup Service [23] or JNDI [21], services can specify and control the QoS - which was not possible with pure JMS.

3 Detailed Solution

The design of JILM will be illustrated via client (Fig. 2) and server (Fig. 3) collaborations.

Client Invocation Layer Initialization. The client uses a lookup mechanism to retrieve a service proxy with the desired interface(s). The service proxy contains a Forwarder instance (ref. Forwarder-Receiver pattern [3]) that encapsulates a `JMSConnectionFactory` and the JMS properties to be used. It creates a connection and registers itself as a JMS message producer.

To receive the results of a method invocation, the Forwarder also creates a Receiver which creates a temporary destination for responses. A temporary destination exists as long as the JMS connection exists. The identifier of the response destination is sent via the `JMSReplyTo` header property of a JMS message.

For time-independent responses, the Receiver within the service proxy transparently makes a durable subscription to an alternative response topic. In order to receive only the client's response messages, a JMS message selector with a permanent client identifier is registered. This approach is also used to store responses when a JMS connection has been lost.

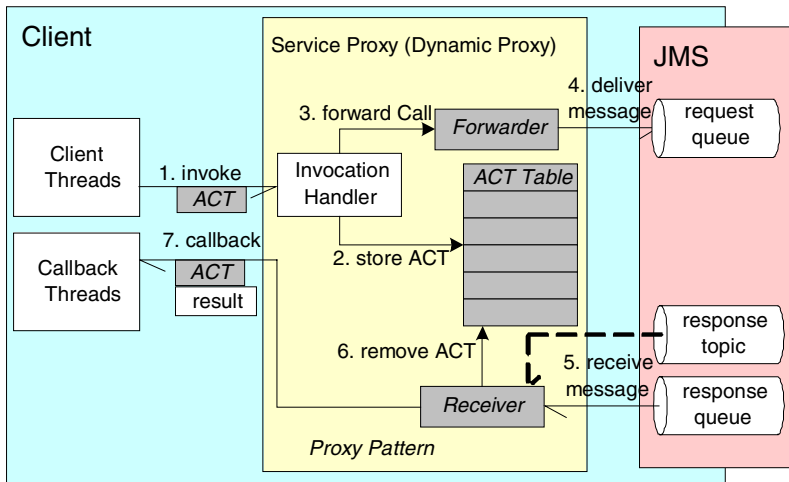


Fig. 2. Client invocation layer

Client-Side Invocation Collaborations. Fig. 2 illustrates the dynamic collaborations among participants in the client.

For an asynchronous invocation, the client explicitly creates and passes an Asynchronous Completion Token (ACT) [20], realized as a `ResponseHandler`, with an invocation (1). The `Invocation Handler` of the service proxy stores the ACT in the ACT Table (2) of outstanding requests and passes the call to the `Forwarder` (3). The `Forwarder` marshals the call arguments (service identifier, method identifier, and method arguments) and the ACT in a JMS message and delivers it asynchronously (4). The identifier of the response destination is sent via the `JMSReplyTo` header property of a JMS message, at which point control is returned to the client invoker.

With a synchronous invocation, the ACT is created internally by the service proxy and the thread is blocked until the response containing the desired ACT is returned.

When the response containing the ACT and the final result is returned (5), the `Receiver` demarshals the JMS message and removes the original ACT from the ACT Table (6). Then a callback thread notifies the client asynchronously about the result by utilizing the `ResponseHandler`'s callback method (7).

To indicate interest in multiple responses, the client supplies a special ACT, realized as a `MultiResponseHandler` type, which is not removed until a message with a completion flag is received.

For futures, the client provides a special ACT, realized as a `FutureResponseHandler` type, where the result is stored until retrieved by the client thread.

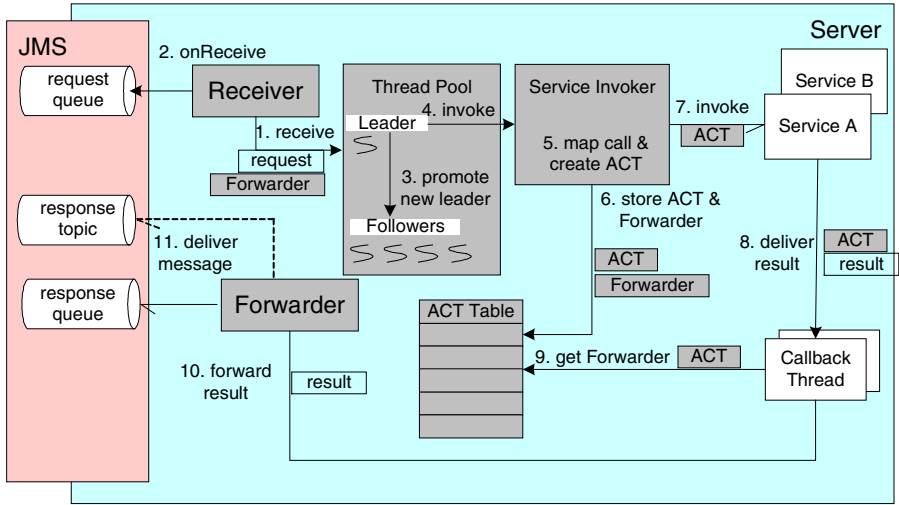


Fig. 3. Server invocation layer

Server Invocation Layer Initialization. A service registers with the invocation layer, which creates and assures distribution of a service proxy to clients. The method mapping table, which maps the service proxy interfaces onto the actual service implementation, is stored in an instance of a service invoker component. A Receiver is created that establishes a connection to the JMS server for a JMS message consumer at the pre-configured JMS message destination. It is responsible for demarshalling messages and maintains references to the service invokers to dispatch the incoming requests.

The Leader/Followers pattern [20] is utilized, where a Thread Pool is initialized and the leader thread calls the receive method of the Receiver (1), which itself invokes the synchronous receive method of the JMS message consumer (2).

Server-Side Invocation Collaborations. Fig. 3 illustrates the dynamic collaborations among participants in the server asynchronous invocation model.

When the JMS message arrives, the Receiver first demarshals the message. Since the message will be discarded, a Forwarder is created (if one does not already exist for this client) which contains the client JMS response destination. Since JMS does not support concurrent sends to one queue, only one Forwarder instance per client destination is created.

Based on the service identifier transmitted with the request message, the Receiver obtains the associated service invoker component. Then the invocation arguments (method identifier, call arguments, and the service identifier reference) along with the Forwarder are passed to the leader thread. This thread immediately promotes a new leader (3) and then becomes a processing thread invoking the service invoker (4). The service invoker creates a new ACT, realized as a ResponseHandler, which identifies the client response Forwarder (5) and is stored in the ACT table (6). The service invoker maps the call onto the actual implementation method and makes the

invocation on the service (7). For an asynchronous method, a reference to the `ResponseHandler` is passed as the first parameter. Before method completion, an incomplete status can be set in the `ResponseHandler`, which will cause it to be retained in the ACT table for future retrieval, e.g. on completion of an asynchronous invocation to another service. The `ResponseHandler` can also be used to cause partial results to be sent to client. When the service has finished processing, the thread returns to the thread pool and becomes a follower thread.

A callback thread from a separate pool is used to send the result, thus decoupling service processing threads from communication issues. The callback thread retrieves the `ResponseHandler` from a queue, which contains the result along with the original client ACT of the invocation (8), and obtains the associated response `Forwarder` from the ACT table (9). Then it invokes the `Forwarder`'s send method (10), which marshals and sends the result with the client ACT in a JMS message (11). Since JMS does not support concurrent message sending, the `Forwarder`'s send mechanism is synchronized.

If the client destination is temporarily unreachable by the service, the `Forwarder` can be configured to either discard results or include an alternative JMS destination (such as a topic) to hold the responses until the client is available. The `Forwarder` must include a unique client identifier with the message to allow the client to select its responses.

4 Performance Evaluation

In this section we evaluate the performance and scalability of our JILM implementation, comparing the throughput of JILM to Java RMI two-way invocations and measuring the overhead of JILM versus a standalone JMS solution. For our measurements we utilized the Sun reference implementation for Java RMI, JDK 1.4.1_01 and the open source project `OpenJMS v.0.7.5` [18] as a JMS implementation. Other JMS providers may perform differently.

JILM Overhead vs. JMS. We first determined the overhead of JILM itself by measuring the average throughput of 10,000 two-way invocations in a single-threaded client application using the asynchronous invocation model of JILM. For this experiment we utilized a 1.7 GHz computer with 768 GB of RAM running Windows XP.

The result was an average additional overhead of 900 microseconds per two-way invocation across both the client and service-side invocation layers versus the time used by JMS.

JILM vs. RMI (Two-Tier). To test the scalability of JILM clients, in this two-tier experiment we compared the throughput of 10,000 two-way invocations in a single-threaded client application using RMI, the asynchronous invocation model of JILM, and the synchronous invocation model of JILM. In order to simulate different call durations, the client invokes a simple method that takes a wait argument specifying a specific server delay before the method returns with a short sample string.

For this experiment we utilized a 1.53 GHz computer with 768 GB of RAM as client and a 1.7 GHz computer with 768 GB of RAM as server. Both were running Windows XP and were connected by a 10 Mbps Ethernet.

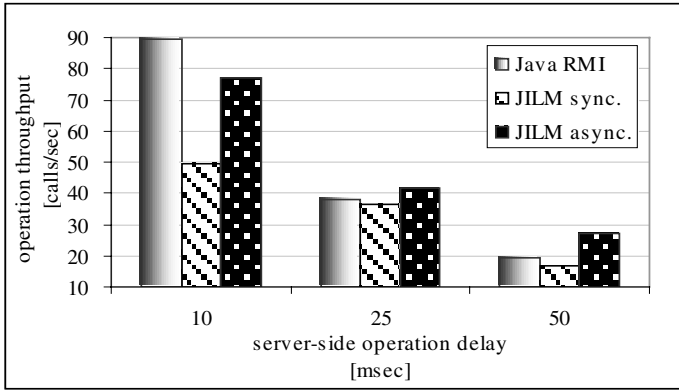


Fig. 4. Two-tier scalability results

Fig. 4 compares the empirical results. For short duration invocations, RMI has better performance than asynchronous JILM due to the overhead of the JILM+JMS implementations. This overhead is also apparent for synchronous JILM, whereby for longer duration calls it nears RMI's throughput.

Thus, once the server invocation delay exceeds the JILM+JMS overhead, the scalability advantages of asynchronous JILM become apparent. In addition, at those invocation delays synchronous JILM can be used to benefit from its simplicity and completability advantages with near RMI performance.

JILM vs. RMI (Three-Tier). In our next experiment, we show the effects of service chaining on scalability. We compared the throughput of 10,000 two-way invocations which were performed by a single-threaded client application using RMI and the asynchronous invocation model of JILM.

For this experiment the client application runs on a 1.53 GHz computer with 768 GB of RAM, the middle tier service on a 1.7 GHz computer with 768 GB of RAM and the sink server on a 1GHz computer with 1GB of RAM. During the measurements the server delay on the middle tier was set to zero and on the sink server it was set to 10, 25, and 50msec.

Fig. 5 compares the empirical results. As in the first experiment, RMI has better performance for short duration invocations. However, as the server delay of the sink server increases, JILM is able to take advantage of asynchrony to achieve higher invocation throughput between the participants in comparison to RMI's blocked calls.

From our three-tier results we see that the use of asynchronous invocations is beneficial for throughput and scalability for cases where service chaining or similar effects occur.

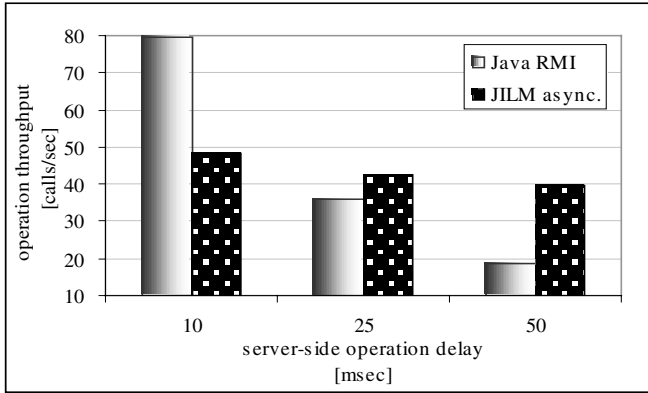


Fig. 5. Three-tier scalability results

5 Related Work

Table 1 shows a comparison of middleware based on our required set of features.

RMI. While callbacks have been used to support a form of asynchronicity within the constraints of the RMI implementation, a number of issues occur:

- clients must make an RMI server available (requires additional client resources),
- server threads may be blocked (e.g. on the callback for busy clients),
- client security issues (with ServerSockets) and firewall issues,
- correlating the client thread context with the response,
- client threads are blocked until the server responds to the initial JRMP call,
- concurrent calls utilize additional sockets.

So while callbacks reduce the client call blocking time equivalent to the server processing duration, as the system scales to capacity, these limitations will affect the system non-optimally.

RMI Variants. In contrast to callbacks, Futures [24], Batched Futures [2], and Promises [11] address call latency by delayed referencing of the result, thus retaining the client thread context from request to response.

Various asynchronous JRMP-compatible RMI variants have been created, such as ARMI [19], Active-RMI [8], and as described by Kerry Falkner [10]. They rely on a modified RMI stub compiler to insert asynchronous behavior and create a separate thread for each invocation. E.g., whereas [10] is an asynchronous RMI implementation that uses Futures, JILM supports Futures without adding Java keywords to the language and without creating a new thread per outstanding request.

JR [9] extends the language to provide a concurrency model on top of RMI, but still has the underlying RMI limitations (although JR is capable of using another transport).

Table 1. Middleware comparison

Feature	JMS	RMI	Async RMI	CORBA AMI	JILM
Object-oriented invocations	N	Y	Y	Y	Y
Synchronous	Y	Y	Y	Y	Y
Asynchronous	Y	N	Y	Y	Y
Time-independent	Y	N	N	Y	Y
Location-independent	Y	N	N ²	Y ³	Y
Group communication	Y	N	N ²	Y ⁴	Y
Batch communication	Y	N	N	N	Y
Interim/partial results	Y	N	N	N	Y
QoS ⁵	Y	N	N	Y	Y
Invocation model decoupling	N/A	N	N	Y ⁶	Y
Provider exchangeability	Y	N	N	N ⁷	Y

Custom-protocol variants that support asynchronous communication include NinjaRMI [15], which requires language extensions. It uses one-way asynchronous communication that may include callback information.

Although RMI does not support group method invocation, much work has been done to create extensions to RMI to support parallel programming environments, e.g. Manta [14], GMI [12], and CCJ [13]. However, our intent and usage model is not parallelism *per se*, and these specialized extensions were unsuitable for use in our context, not fully supporting model decoupling, QoS, and provider exchangeability.

CORBA AMI. While CORBA AMI provides asynchronicity, it requires an IOR or IOGR in order to make an invocation [6], limiting its LII capability. On the other hand, JILM does not require a valid service reference, but can store the invocation in JMS until a service instance exists and retrieves it. Neither batch invocations nor partial or interim results are supported.

While CORBA Messaging implementations could be exchanged, the middleware protocol is specified, whereas JMS does not specify the protocols used, thus JILM

² Possibly supported by parallel RMI extensions.

³ With the addition of FT CORBA [16].

⁴ With the use of Data Parallel CORBA [17].

⁵ E.g., ordering, prioritization, delivery guarantees, durability, rerouteability, etc.

⁶ Not included in the specification, but demonstrated in [4].

⁷ Implementations can be exchanged, but the protocol is fixed.

provides a greater degree of provider flexibility. For Java environments, Java ORB implementations that fully support the AMI specification are not currently known to us or in wide use at this time.

Deshpande [4] describes an asynchronous method handling (AMH) mechanism for decoupling client invocations from server invocations, in particular for middle-tier servers. JILM's service invocation layer supports an equivalent capability for Java services while supporting interchangeability of JMS providers or other middleware via the Forwarder-Receiver.

6 Conclusion

RMI has not addressed the demands in today's systems for asynchronicity and remote invocation completability. In particular, it lacks the desired properties as shown in Table 1.

While JMS has been used in distributed computing to address some of these issues, its usage entails issues including the lack of an object-oriented invocation model, implicit contracts and coupling, and weak typing. This is the area that JILM addresses. By providing an invocation layer around JMS, we were able to mitigate many issues related to JMS, while supporting the desired (a)synchronous invocation models and completability properties.

Our performance measurements show that JILM adds 900 microseconds in roundtrip overhead to the JMS implementation. The asynchronous JILM scales well as the call duration increases. Our three-tier results showed that an asynchronous invocation model is beneficial for throughput and scalability for cases where service chaining or similar effects occur. In addition, the JILM synchronous model can be used for simplicity and yet achieve near RMI throughput for longer server call durations while benefiting from JILM's completability advantages.

By relying on patterns for JILM, our design can be readily applied and reused on various platforms. For systems or architectures that are considering JMS usage, JILM provides an easier and higher-level programming model than direct messaging. In addition, the use of object-oriented invocations ensures that the client utilizes the types and methods expected by the service and allows the service to provide a client proxy that encapsulates the communication mechanism expected by the service. One-way calls could easily be supported if desired.

JILM addresses both the need for asynchronous invocations in Java and the need for remote invocation completability assurances in today's systems.

References

1. Arulanthu, A. B., O'Ryan, C., Schmidt, D.C., Kircher, M., Parsons, J.: The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging. In Proceedings of the IFIP/ACM Middleware 2000 Conference (2000)
2. Bogle, P., Liskov, B.: Reducing Cross Domain Call Overhead Using Batched Futures. In Proc. OOPSLA'94, ACM SIGPLAN Notices, volume 29 (1994)

3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley and Sons Ltd. (1996)
4. Deshpande, M., Schmidt, D.C., O'Ryan, C., and Brunsch, D.: The Design and Performance of Asynchronous Method Handling for CORBA. In *Proceedings of the Distributed Objects and Applications (DOA) conference* (2002)
5. Eckerson, W.W.: Three Tier Client/Server Architecture: Achieving Scalability, Performance and Efficiency in Client Server Applications. In *Open Information Systems*, vol. 10 (1995)
6. Gore, P., Cytron, R., Schmidt, D., O'Ryan, C.: Designing and Optimizing a Scalable CORBA Notification Service. In *Proceedings of the ACM SIGPLAN workshop on languages, compilers and tools for embedded systems* (2001) 196–204
7. JavaGroups Web Site: <http://www.javagroups.com>
8. Karaorman, M., Bruno, J.: Active-RMI: Active Remote Method Invocation System for Distributed Computing using Active Java Objects. In *TOOLS USA* (1998)
9. Keen, A., Ge, T., Maris, J., Olsson, R.: JR: Flexible Distributed Programming in an Extended Java. In *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems* (2001)
10. Kerry Falkner, K.E., Coddington, P.D., Oudshoorn, M.J.: *Implementing Asynchronous Remote Method Invocation in Java*. University of Adelaide (1999)
11. Liskov, B., Shriram, L.: Promises: Linguistic Support for Efficient Asynchronous Procedure Calls in Distributed Systems. In *Proc. SIGPLAN'88 Conf. Programming Language Design and Implementation* (1988) 260–267
12. Maassen, J., Kielmann, T., Bal, H.E.: GMI: Flexible and Efficient Group Method Invocation for Parallel Programming. In *LCR-02: Sixth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers. Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg New York (2002)
13. Nelisse, A., Maassen, J., Kielmann, T., Bal, H.E.: CCJ: Object-based Message Passing and Collective Communication in Java. In *Concurrency and Computation: Practice and Experience*, Vol. 15, Issue 3-5 (2003) 341–369
14. van Nieuwpoort, R., Maassen, J., Bal, H., Kielmann, T., Veldema, R.: Wide-area parallel computing in Java. In *Proc. ACM 1999 Java Grande Conference* (1999) 8–14
15. NinjaRMI Web Site: <http://www.eecs.harvard.edu/~mdw/proj/old/ninja/index.html>
16. Object Management Group: *Common Object Request Broker Architecture (CORBA) Core Specification*, 3.0 ed.
17. Object Management Group: *Data Parallel CORBA Specification*. (May 1, 2002)
18. OpenJMS Web Site <http://openjms.sourceforge.net/>
19. Raje, R., Williams, J., Boyles, M.: An Asynchronous Remote Method Invocation (ARMI) Mechanism for Java. In *Proc. of the ACM 1997 Workshop on Java for Science and Engineering Computation* (1997)
20. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. Vol. 2. Wiley and Sons Ltd. (2000)
21. Sun Microsystems: *Java 2 Platform Standard Edition version 1.4.1*
22. Sun Microsystems: *Java Message Service Specification*, version 1.1 (2002)
23. Sun Microsystems: *Jini Architecture Specification*, version 1.2 (2001)
24. Walker, E. F., Floyd, R., Neves, P.: Asynchronous Remote Operation Execution In Distributed Systems. In *Proc. of the Tenth International Conference on Distributed Computing Systems* (1990)

Web Services-Based Transactional Workflows – Advanced Transaction Concepts

Kai Güntzel

Fraunhofer Institut für Arbeitswirtschaft und Organisation, Nobelstr. 12,
70569 Stuttgart, Germany
Kai.Guentzel@iao.fraunhofer.de

Abstract. Web Services provide a suitable technical foundation for making business processes accessible within and across enterprises. The business logic encapsulated inside Web Services resides often in already existing transactional backend-systems. However, the scope of these systems is normally limited to their domain and is not distributed across heterogeneous environments. In this paper, we look at the impact of the emerging Web Service technology on transactional backend-systems and how transactional contexts needed to propagate activities or even business processes to other services and Service Requestors can be negotiated automatically due to policies attached to the corresponding Web Service descriptions. If error recovery is needed, compensation of already committed Web Services encapsulating activities or business processes in distributed transactional backend-systems can be achieved due to the policy-based selection of the compensating Web Services matching the business and operational properties resulting in a new form of distributed computing.

1 Introduction

Web Services are considered as solution for solving the interoperability problem and the challenge of integration. Before discussing why Web Services may be seen as interoperable integration paradigm, we explain in chapter 2 our understanding of the notion of a Web Service, take a quick tour over some distributed computing concepts and sketch the potential of code-reuse and the resulting impacts at the Web Service-level. We discuss distributed transactions in a general manner in section 3 and lead over to the two common aggregation models in the world of Web Services: BPEL4WS, Business Process Execution Language for Web Services, and Web Services Coordination. Web Services, whether with transactional or nontransactional implementations, can be mixed within an aggregation model. If this aggregation has to be aborted or cancelled, compensation is key.

A possible solution for automatic compensation of distributed transactions encapsulated in Web Services and consumed by BPEL processes is presented in section 4: we show how transactional handshake between a Service Requestor and a Service Provider can be realised based on policies, resulting in a well-defined compensational behaviour, ensuring a consistent outcome, even if the executed process has to be cancelled. Finally, in chapter 5, we give a short example where the requested Web Services are provided by SAP R/3 as transactional functions, often without own transaction boundaries. We conclude with a summary and give an outlook on future research.

2 Web Services – A New Distributed Computing Architecture

A Web Service can be considered as a virtual software component that hides middle-ware idiosyncracies like the underlying component model, invocation protocol etc. as far as possible [1]. The business logic encapsulated in Web Services can only be accessed via well-defined interfaces therefore resulting in opaque components. The consumer of a Web Service normally doesn't know anything about the implementation details, the specifics of the underlying runtime environment and so on. Whereas in B2C scenarios the user interacts via an internet browser, Web Services are for B2B and not meant for direct user-interaction. However, this doesn't count out that the results or HTML-pages the user consumes are generated due to the outcome of Web Services.

2.1 The Story of Distributed Computing

Over the past decades, several distributed computing architectures have been invented, amongst them the Remote Procedure Call (RPC) [2], the Common Object Request Broker Architecture (CORBA) [3], the Distributed Component Object Model (DCOM) [4] and not to mention all flavours of messaging middleware. Despite their intended interoperability approaches, none of them reached broad pervasiveness whereupon the reasons were multifaceted: the classical RPC became widespread in UNIX-environments but it was very hard to get UNIX- and Windows-machines connected. A better but at the same time much more complex architecture was invented by the OMG with CORBA. Here, interoperability was made possible by providing Object Request Brokers (ORBs) for different computing platforms. Nevertheless, practice was much more complicated: explicitly generating Stubs for each programming language and missing or incompatible ORBs for some platforms turned out to be too complex or too expensive for integrating heterogeneous systems. In the meantime, the Windows environment has been extended by the COM-architecture to support communication between remote Windows-systems. The only platform-neutral and therefore interoperable solution was adopted from Mainframe-environments: messaging (message queuing, to be more precise).

This concept was beared in mind when the Simple Object Access Protocol (SOAP) [5] was developed: a simple XML-based communication protocol for messaging and RPCs. Due to its XML-based format, SOAP-processors can be easily developed for a huge range of systems. The ability to easily represent data in XML eliminates byte ordering concerns to describe these data structures and to validate whether data coming on the wire complies with its specification [6].

Yet another aspect which contributed to the success of SOAP was the idea not to define a new transport protocol but to use existing transports, such as HTTP, SMTP, or JMS [7]. The benefits of HTTP are obvious: since the pervasion of the Internet, HTTP is available on (nearly) every platform and therefore facilitates interoperability and last but not least it is much easier to pass firewalls with a well know protocol like HTTP than e.g. with IIOP.

2.2 The Service Oriented Architecture

Web Services are provided ($\rightarrow$ *publish*) by Service Providers, discovered using a Service Directory ($\rightarrow$ *find*) and bound dynamically by Service Requestors ($\rightarrow$ *bind*). All the action is based on standards-based protocols and formats in order that interoperability is achieved, even when the partners are using heterogeneous hard- and software, and thus resulting in a real interoperability approach. This service oriented approach therefore switches from tightly coupled applications to loosely coupled services.

The transport medium used by this Service Oriented Architecture (SOA) is (normally) based on TCP/IP and the data and methods exchanged by the involved partners are - as already mentioned - encoded in XML. Web Services can be considered as firewall-compliant RPCs, since the standard transport protocol used by SOAP is HTTP. The Web Service invocation is packed in the payload of a SOAP-message (to be precise in the SOAP body of the SOAP envelope which forms the SOAP payload) and sent to a SOAP Endpoint Reference, the Service Provider.

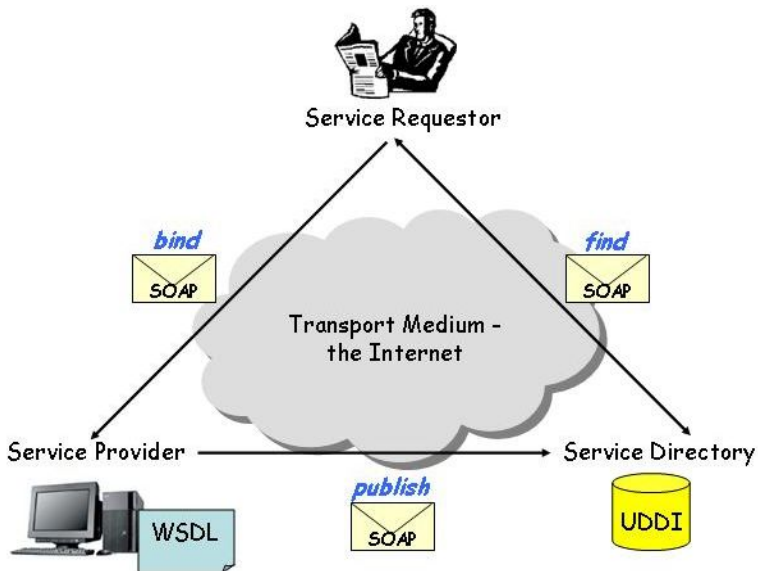


Fig. 1. The Service Oriented Architecture

Web Services can be discovered by browsing a Service Directory, the so called Universal Description, Discovery and Integration or for short UDDI [8]. Service Providers are populating UDDI with information about the Web Services provided by them. This information contains amongst technical descriptions (which methods can be called, what are the parameters etc.) information about the Service Provider himself, i.e. some background about the person providing the service.

The functionality provided by Web Services is described using the Web Service Description Language (WSDL) [9], defining Web Services as collections of communication end points that can exchange certain messages. In other words, a WSDL document describes a Web Service's interface and provides a network address where the

Web Service could be found. To be more precise, a complete WSDL service description provides two pieces of information: an abstract service description and the specific protocol-dependent details that users must follow to access the service at concrete service end points [10], the concrete binding information. Figure 2 shows the abstract description of a Web Service as part of the service description in terms of messages exchanged in a request-response interaction. These messages provide an abstract typed data definition sent to and from the involved participants, i.e. Web Services.



Fig. 2. WSDL part 1: the abstract description of a Web Service

The operation and portType elements combine the existing messages to define the interaction between the Service Provider and the Service Requestor, whereas each operation represents a message exchange pattern that the concrete Web Service supports. Therefore, an operation is a combination of messages marked as input or output (a fault label exists also, but is not shown in our example), indicating what role is played by the message sent or received in a given interaction.

The portType in turn is a collection of operations that are all supported by the specified end point (see later).

To complete the service description we finally have to provide the necessary information *what* communication protocol to use, e.g. SOAP over HTTP, *how* to accomplish individual service interactions over this communication protocol and *where* to find the specified service. This is done with the WSDL's concrete binding information, see figure 3. For a deeper insight in WSDL have a look at [6], [9] or [10].

Recapitulating, one has to clarify that Web Service technology is about *how* to access an application – it is not an implementation technology!

```

<binding name="POSoapBinding"
  type="tns:POPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetPriceInfo"
    <soap:operation style="RPC"
      soapAction="http://water-level.com/priceInfo"/>
    <input>
      ...
    </input>
    <output>
      ...
    </output>
  </operation>
</binding>

<service name="shopservice">
  <port name="shopservicePort" binding="tns:POSoapBinding">
    <soap:address location="http://water-level.com/shopservice"/>
  </port>
</service>

```

Fig. 3. WSDL part 2: the Web Service's concrete binding information

2.3 Code Reuse at the Web Services-Level

Often, the functionality provided by the Web Service component is already in place and in that case “encapsulated” in WSDL to provide the same features (i.e. methods, functions or procedures) as in the originating backend-system or perhaps a less complex dependent to the unpredictable community of Service Requestors. The provided Web Service can be consumed “lonely” or be the activity implementation of a more complex business process, consisting beneath this specific Web Service of other activity implementations which can be Web Services too or any other possible activity implementation as e.g. reports, queries or even interactive dialogs.

Anyway, the Web Services' inherent business logic often incorporate some transactional characteristics – whether the action triggered by a Web Service has to be bracketed in transaction boundaries (as it is the case with transactional functions) or the Service Requestor has to deal himself about erroneous situations, i.e. has to repair pieces of work done by the Web Services' implementation.

Like classical RPC-environments, the Service Oriented Architecture therefore provides a distributed computing environment with Web Services as resources – and in the same way has to cope with the problems of reaching consistent outcome in distributed applications, especially when different resource managers get involved. Besides the fact that HTTP as the preferred transport medium for Web Services is stateless it gets hard to find an agreement on how to preserve transactional characteristics of remote systems, notably when the involved partners encapsulate long-running transactions. This is not a special feature regarding Web Services but the developed

solutions for the integration of distributed transactions cannot be adopted to the Web Services-world without modifications and careful considerations.

3 Distributed Transactions and Aggregation Models in the World of Web Services

The fundamental concept of transactions defined by the ACID paradigm [11] can't be adopted without modifications to distributed environments, especially if high throughput and concurrency between distributed transactions, spanning several different resource managers, are requested. The classical Two-Phase Commit Protocol [12] ensures that all involved resource managers reach consistent outcome on behalf of a distributed transaction. This works well as long as all operations are performed under control of a single application which brackets the transaction. Instead, this concept lacks, when one wants to control several operations, each under the control of its own transaction manager with its own transaction boundaries.

If we take an application like a Workflow Management System, coordinating the execution of the various activities constituting a single business process, the corresponding activities are no longer independent of each other: the failure of one activity might impact other activities. This gets even more complicated, if the involved activities aren't simply executed in a consecutive manner but are spanning a tree of dependencies or are workflows themselves.

A lot of good concepts and ideas have been elaborated over the past years to provide (nearly) transactional behaviour for these cases. We will shortly outline some of these concepts before introducing two important aggregation models for Web Service environments.

A nested transaction representing a collection of transactions with a tree structure is isolated from all other (nested) transactions therefore this instance of distributed transactions is also called closed nested transaction model. Because of the ensured isolation property concurrent access to shared data is dramatically reduced resulting in a diminished overall throughput [13].

Giving up the isolation property, i.e. releasing the locks before the final commit or abort, while retaining intra-transaction parallelism promises to increase the throughput within the overall environment [13]. To achieve this, the nested transaction model is modified to that effect that the outcome of a transaction is made visible at commit to all other transactions of the environment and not just to the parent within its embracing nested transaction. This internal violation of the isolation property implicates that in case of failures or aborts already committed transactions have to be compensated by some application-specific functions, the so called compensation functions.

This leads directly to the concept of compensation spheres. A compensation sphere is a collection of activities with transactional or nontransactional implementations which are tightly related [13]. If one of these pieces of work has not performed correctly and has to be aborted or repaired, all the other pieces of work that have already performed have to be repaired, i.e. undone, too. This work is done by compensation actions – the activities inside a compensation sphere are therefore actually pairs of activities: the

intended activity and the inverse activity, undoing the work in case of failures which has been performed so far by its sibling.

3.1 Aggregation Models for Web Services

The so far introduced set of Web Service specifications (WSDL, SOAP, UDDI) defines protocols for Web Service interoperability but lacks protocols for choreography. This becomes more and more important as with Web Services it is easy to tie together a large number of participants forming large distributed applications. The resulting business processes can be complex in structure, with complex relationships between their participants.

In the last year, two aggregation models for the world of Web Services have been introduced amongst others by an industry-consortium. Both of them have specific use cases which will be shortly sketched as these concepts will be the foundation of our transactional handshake introduced later in this paper.

3.2 Business Process Execution Language for Web Services

The Business Process Execution Language for Web Services (BPEL for short) [14], in the meantime submitted to OASIS [15], is an aggregation model for Web Services, whereas the resulting business process in turn can be considered as a composite service, i.e. a Web Service. The aggregation is done at the portType level by specifying both, the portType required by the aggregate as well as the portType offered by the composite service, e.g. pT_{CS} in figure 4. BPEL is a choreography centric approach, i.e. it prescribes the potential order in which the operations of the aggregated portTypes are to be used [16].

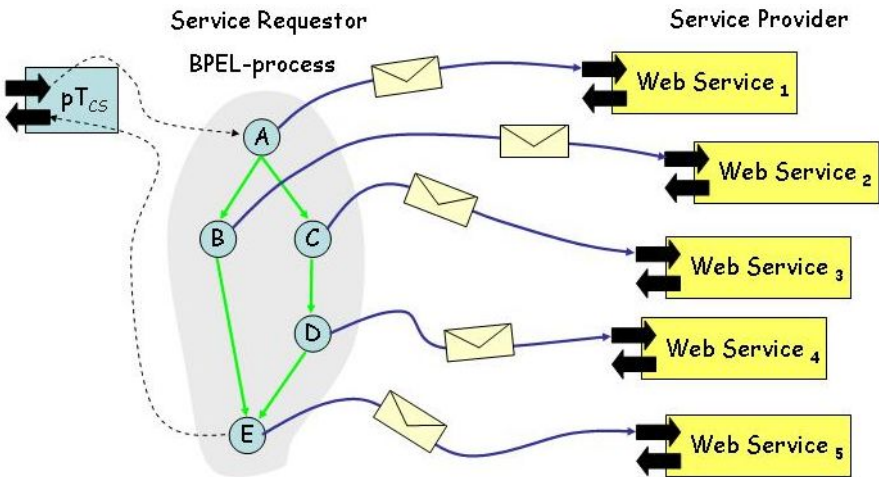


Fig. 4. An example of aggregated Web Services forming a BPEL-process

3.3 Web Services Coordination

Together with the first BPEL-specification, the same consortium introduced also a specification describing an extensible framework for providing protocols that coordinate the actions of distributed applications: Web Services Coordination (WS-C) [17]. The protocols (see below) defined upon this framework enable participants to reach consistent agreement on the outcome of distributed applications with the aid of one or more Coordinators whereas the use of this coordination framework is not restricted to transaction processing systems!

Here, the aggregation takes place at the instance level of the involved services which don't have to be executed in a prescribed respective ordered manner.

3.4 Web Services Transaction

A third specification, Web Services Transaction (WS-Tx) [18], describes coordination types that are used with the coordination framework. A coordination type can have several coordination protocols, each intended to coordinate a different role that a Web Service plays in the activity. WS-Tx provides in its specification two coordination types including their respective protocols: the Atomic Transaction specification defines protocols that enable existing transaction processing systems to wrap their proprietary protocols and interoperate across different hardware and software vendors. The Business Activity specification defines protocols that enable existing business process and workflow systems to wrap their proprietary mechanisms and interoperate across trust boundaries and different vendor implementations [18].

4 Business Processes in the World of Web Services

Putting these two aggregation models together, a full Web Service-based distributed business process could be realised. Indeed, according to the BPEL-specification, the Web Services i.e. the activity implementations forming the BPEL-process won't be distributed or span multiple vendors and platforms. Therefore, if a BPEL-process wants to consume a remote Web Service in its flow, the desired Web Service has to provide in advance some information about its characteristics, in particular which Web Service could be used to compensate the action taken before in case of failures because error handling in BPEL-processes is limited to the actual domain.

As a process expressed in BPEL can be considered as a Web Service again, this Web Service can be a participant in another BPEL-process or distributed application under the control of WS-Coordination, whether as an Atomic Transaction or more probable as a Business Activity. If the encapsulated BPEL-process is consumed by a second (remote) BPEL-process, the information about a compensating business process must be provided in the same manner as with a single Web Service too.

This seems to be complex, especially the fact, that the BPEL-engine has to remember where to find and finally invoke the compensating Web Service. In the following section we will show how the necessary information regarding the compensating Web

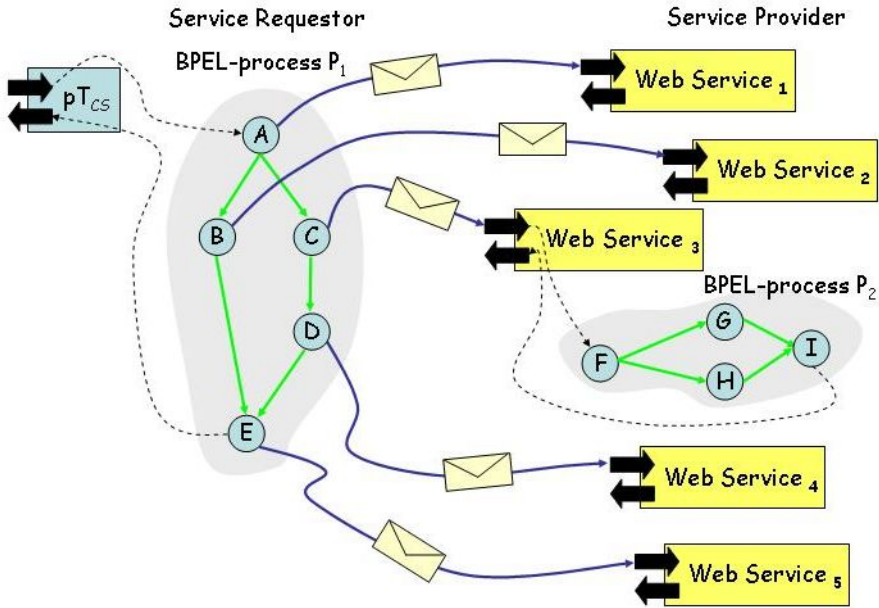


Fig. 5. A BPEL-process consuming another BPEL-process

Service can be published in a general manner and how the information retrieved could be used in a compensation-scenario.

4.1 Web Service Policy

The Web Service Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service [19]. The goal of WS-Policy is to provide the mechanisms needed to enable Web Service Providers to specify their policy information. This information is expressed through an XML-based structure called a policy expression and a core set of grammar elements to indicate how the contained policy assertions apply to a policy subject, i.e. the endpoint or resource to which the policy is bound. These policy information can be either associated with specific instances of WSDL services or be referenced from WSDL definitions [20]. Thus, the Service Provider can expose the conditions under which he provides the Web Service.

4.2 Transactional Handshake between Web Services

The essential precondition to be able to compensate a Web Service WS_i is the definition respective the existence of a compensating Web Service cWS_i undoing the effects caused by WS_i – an obvious but important fact.

The information about the Web Service WS_i and its sibling cWS_i is propagated in a policy-file attached to the Web Service's normal service description expressed in a WSDL-file, see figure 6.

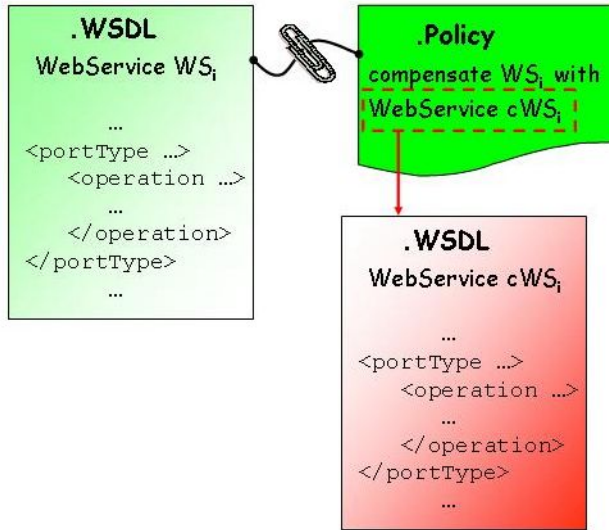


Fig. 6. Service description with attached policy-information

The following figure shows exemplarily how a Service Requestor, in our case the BPEL-process P_1 , subscribes in a “contracting phase” to a provided Web Service WS_3 , which in turn is a composite Web Service, i.e. another BPEL-process P_2 . At the same time, the attached policy-file is inspected, the compensating Web Service cWS_3 is found and the provided portTypes are referenced for the use in the BPEL-process’ compensationHandler.

Consequently, if the BPEL-process has to be cancelled, the process engine rolls back all activities in its flow, invoking P_1 ’s compensation handlers using the compensate activity as shown in figure 8.

Therefore, we realized with policy-files distributed BPEL-processes with opaque compensation behaviour at adjacent process-engines: even though P_2 has terminated after its successful invocation as activity implementation of C respective WS_3 , the effects can be undone thanks to the compensating process cP_2 , linked in the policy-file of WS_3 and defined through its composite service description cWS_3 .

We can go even further and participate with a BPEL-process in distributed applications under the control of WS-Coordination. Here too, the compensating process linked in the appendant policy-file comes into play during the coordination protocol flows namely when the Coordinator requests a rollback.

The policy-information regarding the BPEL-process can be even augmented with information about the supported coordination protocol(s) and appropriate or required Coordinators.

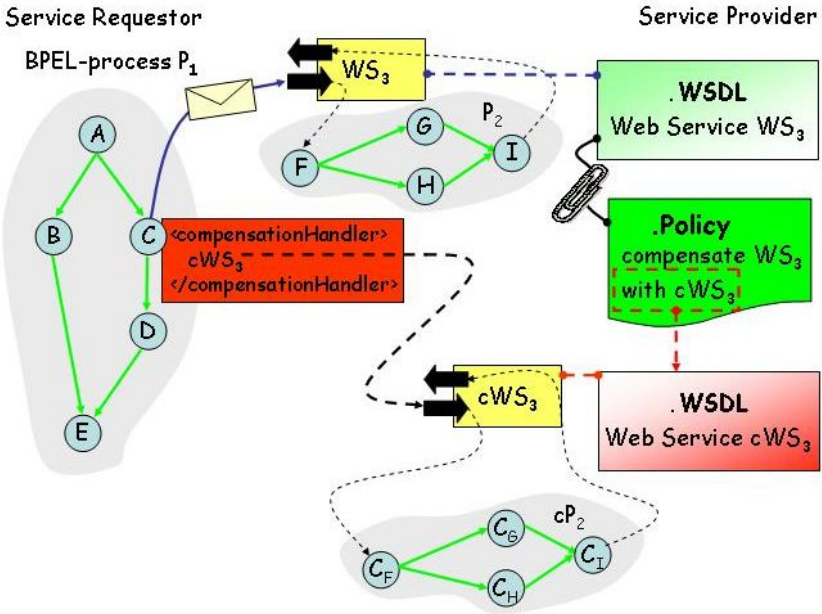


Fig. 7. Contracting Phase: referencing the portTypes of Web Service WS_3 and cWS_3

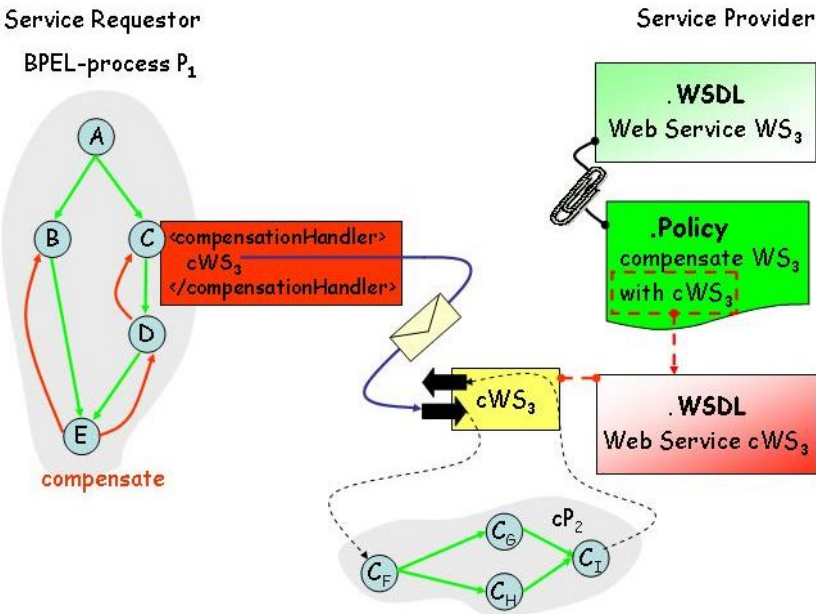


Fig. 8. Opaque compensation of a remote BPEL-process

5 SAP R/3 Functions as Web Services

To conclude, we will deliver a short insight about the cooperation of SAP R/3 [21], a widespread Enterprise Resource Planning system, and the world of Web Services. Thanks to the SAP NetWeaver [22] from SAP (formerly known as mySAP Technology), every functional component respective transaction, e.g. BAPI, RFC, IDOC, in SAP R/3 is available as a Web Service. Here, the user has to pay attention whether the requested R/3-function encapsulated in a Web Service sends an implicit Commit to the server, resulting in the release of all locks and an update in the database.

Otherwise, the user has to explicitly send a `BAPI_Transaction_Commit` to reach a consistent state and ensuring that all data is updated in the database according to the intended outcome. Understandably, one has to tell the R/3-server which transactions have to be committed. On this account, the programmer or process designer has to establish a transaction boundary, bracketing the Logical Unit of Work (LUW), and therefore signals doubtless which Web Services should be transformed into a durable state. The information whether one has to explicitly Commit or not can be found in the policy-file. Certainly, this concept of explicitly signalling the Commit-state of a transaction facilitates the attendance as participant in a coordination type.

6 Summary

In this paper we have first introduced the Service Oriented Architecture as the base for a new paradigm of distributed computing. As we have stated out, Web Service technology is about accessing an application and not an implementation technology. The Web Services paradigm therefore enables the transition from tightly coupled applications to loosely coupled services. With the support of transactions and the composition of Web Services towards complex processes, BPEL4WS could be the right choice when orchestration respective flow aspects become apparent.

We have shown that Web Services have to be compensated in case of failures or cancellation. This will only be possible, if the compensating Web Service is known a priori. The contracting between Service Requestor and Service Provider ensuring an absolutely reliable and at the same time opaque compensational behaviour even across domains could be realised with attached policies.

This implies, that the basic information, whether the Web Service is an Atomic Transaction or a long running Business Activity is also known, thus enabling a transactional handshake between the two parties.

The next step in service oriented computing is Grid Computing. In Grid environments, it will be possible to choose from a wide range of services, which will be hosted by OGSA-compliant [23] Service Providers. If a selected service isn't available any more, it's up to the Grid to fetch the next suitable service. This can be done by virtue of service level agreements which in turn can be based on policies. In addition, one can subscribe to an abstract service and it is the Grid's job to select the best suited service regarding response time, availability, costs etc.

What a wonderful world ... of Web Services!

References

1. Frank Leymann: *Web Services and Business Processes*, Fraunhofer IAO Symposium Collaborative Business, Stuttgart, Germany, July 9, 2002
2. The Open Group: *The OSF Distributed Computing Environment*, <http://www.opengroup.org/dcel/>
3. OMG: *Common Object Request Broker Architecture (CORBA)*, December 6, 2002, http://www.omg.org/technology/documents/formal/corba_iiop.htm
4. Microsoft: *The Distributed Component Object Model*, <http://www.microsoft.com/com/tech/dcom.asp>
5. Martin Gudgin et al.: *SOAP Version 1.2 Part 1: Messaging Framework*, W3C, December 19, 2002, <http://www.w3.org/TR/2002/CR-soap12-part1-20021219/>
6. Steve Graham et al.: *Building Web Services with Java*, Sams Publishing, 2002
7. SUN: *Java Message Service API*, <http://java.sun.com/products/jms/index.html>
8. UDDI.org: *UDDI Version 3.0*, Published Specification, July 19, 2002, <http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf>
9. Eric Christensen et al.: *Web Services Description Language (WSDL) 1.1*, W3C, March 15, 2001, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
10. Francisco Curbera et al.: *Unraveling the Web Services Web – An Introduction to SOAP, WSDL, and UDDI*, in IEEE, 2002
11. Philip A. Bernstein, Eric Newcomer: *Principles of Transaction Processing*, Morgan Kaufmann Publishers, 1997
12. Jim Gray, Andreas Reuter: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1992
13. Frank Leymann, Dieter Roller: *Production Workflow*, Prentice Hall, 2000
14. Tony Andrews et al.: *Business Process Execution Language for Web Services (BPEL4WS) 1.1*, BEA, IBM, Microsoft, SAP, Siebel, March 31, 2003, <ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf>
15. OASIS Web Services Business Process Execution Language TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
16. Frank Leymann: *Distributed Applications without Limits – an Outline*, Keynote at BTW 2003 & KiVS2003, Leipzig, Germany, February 26, 2003
17. Felipe Cabrera et al.: *Web Services Coordination (WS-Coordination) 1.0*, BEA, IBM, Microsoft, August 9, 2002, <http://www-106.ibm.com/developerworks/library/ws-coor/>
18. Felipe Cabrera et al.: *Web Services Transaction (WS-Transaction) 1.0*, BEA, IBM, Microsoft, August 9, 2002, <http://www-06.ibm.com/developerworks/webservices/library/ws-transpec/>
19. Don Box et al.: *Web Services Policy Framework (WS-Policy)*, BEA, IBM, Microsoft, SAP, December 18, 2002, <http://www-106.ibm.com/developerworks/webservices/library/ws-polfram/>
20. Don Box et al.: *Web Services Policy Attachment (WS-PolicyAttachment)*, BEA, IBM, Microsoft, SAP, December 18, 2002, <http://www-106.ibm.com/developerworks/library/ws-polatt/>
21. SAP AG: *SAP R/3 Enterprise*, Walldorf, Germany
22. SAP AG: *SAP NetWeaver*, Walldorf, Germany
23. Ian Foster et al.: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, The Global Grid Forum, the latest Version can be found at <http://www.globus.org/research/papers/ogsa.pdf>

MIDAS's Routing and Scheduling Approach for the Australian Transport Industries

Manish Malhotra and Zahir Tari

School of Computer Science and Information Technology
RMIT University, Melbourne, Australia
{manish, zahirt}@cs.rmit.edu.au

Abstract. Effective and efficient route scheduling can mainly affect client/customer satisfaction and operating costs in the transport industry. Dynamic scheduling has simplified transport logistics such as courier services by providing technology-enhanced, real-time communication. Service requests from the same area should be served once rather than multiple times, facilitating a huge saving in travel distance and time. Mobile Intelligent Distributed Application Software (MIDAS) develops an autonomous routing and scheduling system for Australian transport industry. This system enables smoother running of transportation logistics with efficient and effective operation costs, by combining wireless and Internet technology. This system can receive orders and requests from mobile devices (Palm) and the Internet, and then schedule and forward the orders to the drivers automatically. Autonomous route scheduling includes static and dynamic scheduling to produce an optimal route on digital maps. Static scheduling is used to deal with non-emergency orders that can be scheduled overnight, to output a better solution with sufficient computation time. Dynamic scheduling can also be used to deal with emergency orders that require real-time scheduling within limited time constraints.

1 Introduction

Whether a company takes an order on the web, or through the traditional channels of phone, fax, or EDI, fulfillment remains a challenge in today's faster-paced business climate. MIDAS (Mobile Intelligent Distributed Application Software) solution [16] fulfils customer orders immediately by improving delivery rate by locating precise truck's location and communicating through various mobile devices; resulting in reduction of the inventory and cost, and improvement in efficiency of an organization. MIDAS fulfils orders on time through a help of integrated processes. Starting from taking complete valid orders through Internet or Palm and storing in company's remote database to locating the best suitable truck for fulfilling that order and then sending the orders and new route and schedule to the driver on their wireless device, MIDAS does it all. It also provides an electronic docket system. It not only let the customer track their order anytime on Internet but also accept Proof of Delivery and Order acknowledgement from driver's Palmtop or Wireless phone for storing in company's remote database. MIDAS improves customer service, maximizes profitability, and increase revenue and market share by positioning the supply chain to meet forecast demand, intelligently promising and capturing orders, seamlessly executing and delivering the order, and monitoring the entire fulfillment cycle.

This paper focuses on the techniques and methodology of MIDAS Routing and Scheduling module. A detailed description of the different MIDAS module can be found in [16]. Effective and efficient route scheduling mainly affects the client/customer satisfaction and operating costs in the transport industry. Dynamic scheduling has simplified transport logistics such as courier services by providing technology-enhanced, real-time communication. Service requests from the same area should be served once rather than multiple times, facilitating a huge saving in travel distance and time. However, the time constraints of individual delivery in courier services increase through the complexity of route scheduling in terms of providing good services and minimizing operating costs.

MIDAS provides a complete autonomous route and scheduling system. This system enables a smoother running of Australian transportation logistics with efficient and effective operation costs, by combining wireless and Internet technology. This system receives orders and requests from mobile devices (Palm) and the Internet, and then these orders are scheduled and forwarded to the drivers automatically. Autonomous route scheduling is the foremost concern of this system, which includes static and dynamic scheduling to produce an optimal route. Static scheduling is used to deal with non-emergency orders that can be scheduled overnight, to output a better solution with sufficient computation time. Dynamic scheduling can also be used to deal with emergency orders that require real-time scheduling within limited time constraints. This system also enhances the system operator's functionalities such as the facility of driver tracking and locating the nearest vehicle with digital maps.

The scheduling requirement of MIDAS is a combination of vehicle navigation and order scheduling. Scheduling requires two searches, i.e. local and global. First, a local path between two points has to be made in a local search according to the map. Then, an order of multiple service locations has to be arranged, as global, to match with the time constraint of the services. Due to the need to manage the complexity of navigation and scheduling in MIDAS, the following points need major consideration.

- Vehicles or customers can appear at any point on the map. They may not necessarily be next to or on a road line. Moreover, the system is not scalable to examine every single road for determining which is the closest for the vehicle to start the journey. Thus, we need to figure out how to find the nearest entry point on the road network in an efficient way for these locations and establish a connection between the two points.
- Branches appear at intersections between two roads; two different paths can reach the same destination. It is also not scalable to examine every branch to obtain the best result. Therefore, recognizing the shortest path between source and destination requires an efficient algorithm as well.
- An optimal route of scheduling requires swapping between different locations to fit in the service time constraints. In addition, the larger numbers of locations being served, a better the solution is. The trade-off between fast computation performance and a better solution has to be kept in balance. In order to balance these factors, different algorithms are required for emergency services and non-emergency services.

In addition to the vehicle navigation and scheduling factors, the following challenges on communication and map manipulation need to be worked through.

- Two communication channels have to be aligned and achieved in this system: a communication channel between applications and a communication channel

between the system and the drivers. In the application communication, a standard communication protocol needs to be established and it must be adaptable for both Palm devices and an Internet server. In the driver communication, a simple messaging system is required.

- The fundamental elements of achieving route scheduling are location and distance information. Therefore, map data is the essential component for a realistic scheduling system. An Australian map is vital for this system.
- The supplementary function of displaying a digital map on an operator application requires the capability of map manipulation. This map manipulation includes the ability to change the map view, allocating positions and showing routes on the map.

2 Background and Literature Survey

The focus of this section is on the existing approaches to vehicle routing problems using digital maps and SMS technology. We first introduce the concept of vehicle routing problems with their existing solutions. A mathematical formulation for problem solving and two types of heuristic techniques will be examined in the subsections. Afterwards, a review of digital maps will be conducted and their technology will be described in the second section. Lastly, the SMS will be covered with its accessing methodologies.

Vehicle Routing Problem

In vehicle routing problem (VRP), a fleet of vehicles based at a central depot must service a set number of customers. The vehicle routes intend to service all the customers exactly once with limited capacity and minimum cost. Moreover, the additional "time window" constraints extend the problem as the vehicle routing problem with time window (VRPTW), which specify the earliest and the latest times of each service, and add the complexity of allowable delivery time. There are also several other models for the time window constraints scheduling problem [12], which include the travelling salesman problem (TSPTW), the shortest path problem (SPPTW) and pickup and delivery problems (PDPTW). However, The VRPTW is the most widely discussed and generic representative to our scheduling problem. These problems have been defined as Non-Polynomial hard (NP-hard) [4] and are best solved by using heuristics. Most heuristic search strategies involve finding an initial feasible solution and then improving on that solution using local or global optimisation techniques [3].

Insertion Heuristic

The insertion heuristic was introduced by Solomon [2], who concluded that the insertion heuristic has an excellent performance, compared to savings heuristic, nearest neighbour heuristic and sweep heuristic. The concept of the insertion heuristic assumes a route R , where C_0 is the first customer and C_m is the last customer with their earliest arrival and latest arrival time. The feasibility of inserting a customer into route R is checked by inserting the customer between all the edges in the current route and selecting the edge that has the lowest travel cost. For customer C_i to be inserted between C_0 and C_1 , the insertion feasibility is checked by computing total load, total

travel time, and amount of time that the arrival time of t_1 is pushed forward. Insertion is only warranted if none of the constraints is violated.

Digital Maps

The advantage of digital maps in MIDAS is to provide visual data information to customers, operators and drivers. Compared to text format information, a simple graphical data presentation can represent a hundred lines of words. Apart from the graphical appearance, map data also contains useful information for computing route scheduling in a more dynamic and realistic form. It is not user-friendly to pre-calculate every single distance between any points on a map. The distance is easier to calculate in the execution time with the current positioning by using GPS coordinates. Furthermore, the directions can also be indicated. The digital map aids in the completion of a schedule. It provides an efficient and quality service for the transport industry. Map data is crucial for the functionality of this system. The data file that has been used in this system is shapefile. A shapefile stores non-topological geometry and attribute information for the spatial features in a data set [7], which is defined by Environment System Research Institute, Inc. (ESRI).

SMS

Short Message Service (SMS) is the ability to send and receive text messages to and from mobile telephones. Each message can be up to 160 characters of text in length. The short messages are not sent directly from the sender to the recipient. The undeliverable message will be stored in an SMS centre and continue its attempts for seven days. An SMS message can be sent and received simultaneously with GSM voice, data and fax calls. The utilization of SMS can provide a simple and convenient way of staying in touch with drivers. In Australia, Telstra MobileNet SMS Access Manager provides flexible access to the SMS network infrastructure via a variety of ways, which include wireless access, Short Message Peer to Peer Protocol (SMPP) access and dial-up access [11]. Apart from the major telecommunication carriers in Australia, the SMS access solution is also available from some wireless service providers, such as BlueSkyFrog. BlueSkyFrog smsAccess is a message gateway, which benefits from two different technologies, Component Object Model (COM) and Simple Object Access Protocol (SOAP), and offers a programming interface to access the SMS gateway through the Internet [9]. It provides a lightweight development environment but offers fewer choices.

3 MIDAS Routing and Scheduling

3.1 MIDAS Background

MIDAS is the result of a successful Australian Research Council (ARC) industry grant. MIDAS [16] provides an autonomous delivery management system from client orders to the proof of delivery for the transport industry. To accomplish this, MIDAS utilizes different technologies, including Global Positioning System (GPS), wireless technology (Short Message Service (SMS)/Wireless Application Protocol (WAP)) and the Internet. One of the main tasks of MIDAS is route planning. This provides both static and dynamic scheduling utilizing wireless communication channels to

keep drivers up-to-date with information in real time, when they are off-site. MIDAS also benefits the clients of the transport companies by easily placing and tracing their orders anywhere and anytime.

MIDAS resides in three components: mobile devices, an Internet server and the MIDAS server. These components provide different capabilities that are presently lacking in the existing software of transport companies. In any case, coexistence between MIDAS with the existing software of these transport companies can also be done without dramatic integration changing.

Figure 1 provides an overview of the MIDAS architecture and its sub-components. The architecture of the system is defined on the principle of a Layered Reference Model. The first layer of the architecture is the "User Interface and Services Layer", which act as the point where the users interact with the system. Beside the user interfaces, this layer also contains service modules that handle different type of communications to the lower layer.

The second layer is the "MIDAS Application Server Layer". This layer contains several subsystems, which work together to provide services to the upper layer. The important subsystems in this layer are: a) *Communication Subsystem* is responsible for handling communication with the layer above and consists of different components to handle different types of communication. b) *Order Subsystem* is responsible for handling all requests that are related to orders, such as accepting new orders, and calculating order invoices. The subsystem will be part of the MIDAS server, which resides at the transport companies. c) *Routing and Scheduling Subsystem*: Effective and efficient route scheduling can mainly affect client/customer satisfaction and operating costs in the transport industry. Dynamic scheduling has simplified transport logistics such as courier services by providing technology-enhanced, real-time communication. Service requests from the same area should be serve once rather than multiple times, facilitating a huge saving in travel distance and time. This subsystem is responsible of dynamic routing and scheduling of the daily run sheet of drivers. d) *SMS Subsystem* is responsible for sending SMS messages to the drivers to inform the details of the new order and interacts with the MIDAS server to check the new coming orders. e) *Storage Subsystem* is accountable for providing a means for storage, such as connection to existing DBMS and file management. All other subsystems depend on this system to perform any queries to the database. By having this subsystem, changes to storage will not affect any of the other subsystems, which improve the system flexibility. This subsystem is part of the MIDAS server, which resides at the transport companies. f) *Tracking Subsystem* is responsible for tracking the trucks and drivers using the GPS satellites.

The last layer of this architecture is the "System Layer". This layer represents the existing company's system environment, which includes the company's DBMS and Operating System.

MIDAS aims at designing and implementing server-side components to supply a routing and scheduling solution and to integrate digital maps to provide an efficient delivery solution in real-time transport industry environment. The scope of this system will enable the handling of the following scenario:

"When the MIDAS server receives a valid order from the Internet/PDA, it has to determine what day the order needs to be achieved by. If an order can be fulfilled on the next day, it will be stored in the database, ready to be

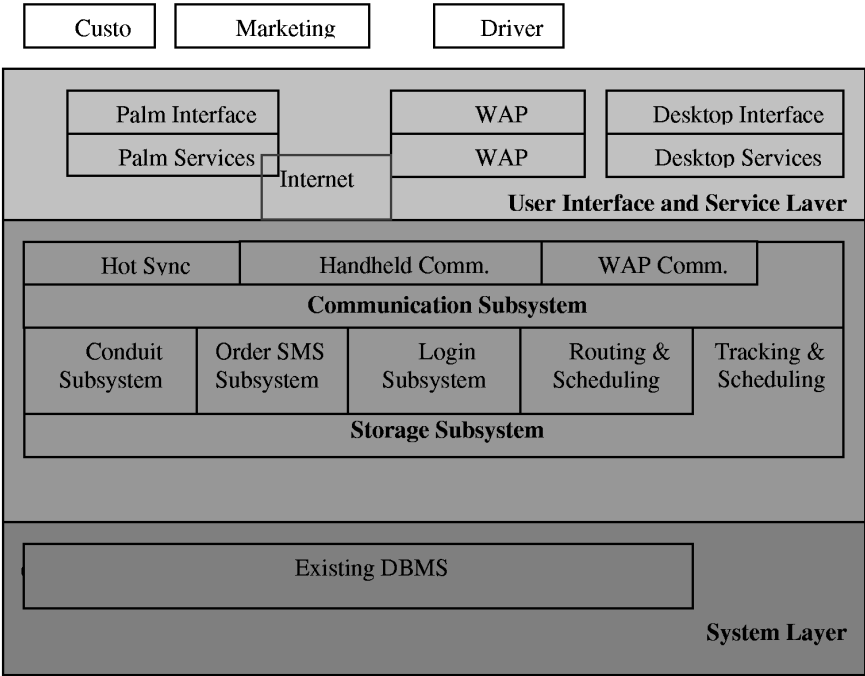


Fig. 1. MIDAS Architecture.

retrieved for scheduling during the nighttimes. Afterwards, the scheduled manifests can be distributed to the drivers in the morning with a supplementary route map. Otherwise, the MIDAS system determines and then requests the most appropriate vehicle that can fulfil the order within the given time frame. The driver of the vehicle is then contacted through Simple Messaging Service (SMS) to accept the new order and new route. Conversely, a customer must be immediately informed about an unresolved order through the autonomous system. From the system operator's perspective, the operator can track all the current locations of the vehicles on the digital map. Furthermore, the closest vehicle can be indicated with a given location. It is also able to show a route on the map for different drivers."

3.2 MIDAS Routing and Scheduling

The main objective of the MIDAS server system is to provide an autonomous dynamic route scheduling system that enables scheduling of transport industry orders in two manners: static and dynamic. Furthermore, the sub-goals of MIDAS need to be accomplished as follows: a scheduling system (which produces optimal routes for the existing order) and a digital map system (which shows the current truck locations and routes on a digital map).

MIDAS Routing and Scheduling module provides the following functionalities: *path searching* (as MIDAS server is able to find an optimal path between two locations); *scheduling* (as MIDAS server is able to schedule any orders); *mapping* (as MIDAS server is able to access map data from a “shape file” and perform mapping); *SMS* (as MIDAS server is able to send a SMS via the Internet; and *database connectivity* (as MIDAS server is able to access a database and answer the query).

These functionalities help the operator of MIDAS to track and trace the truck and driver's current locations. It also helps the operator to find the closest truck to a particular source point and to send the driver a SMS message regarding this.

3.2.1 MIDAS Routing

Routing is a fundamental part of scheduling, which performs a dynamic path searching with map data to produce an optimal route. In this section, we will review our proposed data structure and algorithm for search execution.

Data Structure

Up to this point, we have the map data and API to access the data file and display it on the screen. However, the data in the shape file is not in a searchable format for path searching. The problems had been raised on how are we going to recognize it is a road on the map and are they connected to each other? The most basic information we have “a road is represented in a poly-line, which contains two or more pairs of coordinates in latitude and longitude values”. Therefore, we need to construct a data structure for ease of search.

In our approach, we create a data structure that is based on binary trees. There are two layers of binary trees. Both are sorted on different values, one is according to the latitude value and the other is according to the longitude value. Besides, a node of the top layer binary tree contains the second layer binary tree. Hence, when a coordinate of a road comes in, it will compare with the latitude value on the first tree and get into the second binary tree. After that, the coordinate will be stored in the second tree according to the longitude value. However, there is the same coordinate that belongs to two or more different roads because of connections or intersections between them. Therefore, the nodes of the second tree allow storing duplicated values for different roads. Therefore, a node of the second tree contains a linked list and the final storage of the coordinates is the linked list, which will also link to all the corresponding road information as well (Figure 2). This structure allows us to search an entry according to two values (latitude and longitude) in $\log N$ complexity, where N is the number of entries.

Closest Point

Due to unrestricted locations, vehicles and customers can be anywhere on the map. A virtual path needs to be established between a specific coordinate and a real road. Other than that, a nearest road has to be selected for forming a real path based on the road network.

It is very difficult to put every point of the roads on distance testing for the nearest, especially when the area of the map is getting bigger. In the proposed solution, we utilize the characteristic of coordinates and sort them using the projected data structure, and then we try to find a small range of nominated coordinates, which are possibly next to the specific location, in a semantic way thus the number of

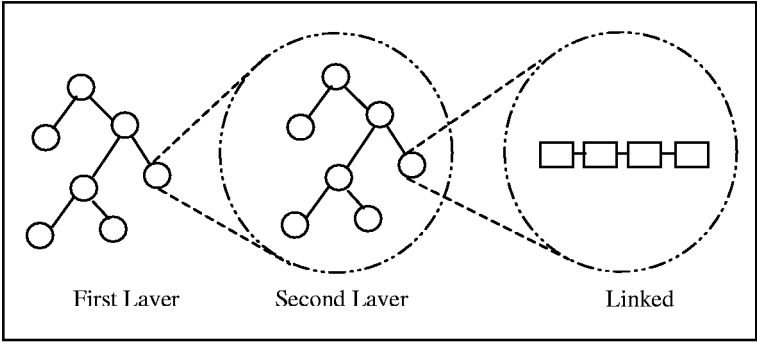


Fig. 2. Double layer tree

comparisons can be reduced. This approach can be also used for searching for the nearest vehicle with a given point.

First at all, four nominated coordinates can be selected based on four directions of the particular coordinate (a centre point, as shown in figure 3 above); the closest point from North, the closest point from East, the closest point from South and the closest point from West. From these four points, we can form a rectangular area, which will cover all the points next to the centre point. However, the rectangle can be very irregular. It may include all the points in a worst case. The further step of improving the nominated area can be done by forming a circular area by using the closest point of these four points. We use the distance between the closest and the centre point as radius to form the circle (figure 3c). As we know, a circle has same distance in all directions at any angle from the centre point; hence, the covered points in this area are

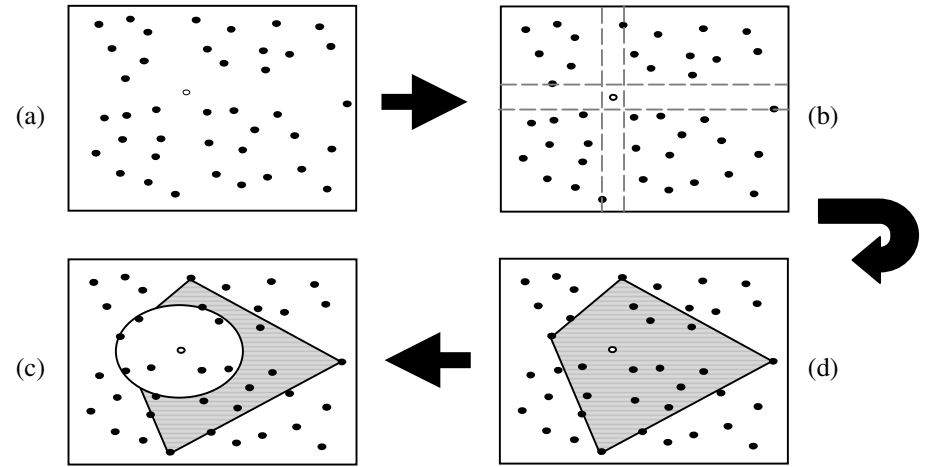


Fig. 3. Closest point.

more representative for the nomination with shorter distances. Furthermore, since the points in the area are much less than the whole map, the number of direct distance comparisons can be reduced significantly. It just uses those points in the nominated area for finding the nearest point as in figure 3d. Hence, time can be saved from comparing all the points. Unfortunately, the circle still may happen to cover all the points in a worst case.

Path Searching

A wise choice on a correct direction while standing in front of an intersection can save a lot of unnecessary traversal for destination reaching. In our approach, we will store all the nominated paths into a limited buffer list, which is sorted according to the approximated distance to the destination in ascending order.

A = an approximated distance

L = the actual path length have found

D = the direct distance from the path to the destination

$$A = \alpha L + \beta D, \text{ where } \alpha + \beta = 1, \alpha \geq 0, \beta \geq 0.$$

Therefore, the list will be resorted when a new branch is added and the first path in the list is always shortest to the destination. The algorithm for path searching is as follows:

```
Begin with the left path and right path of the starting
point and store them into the buffer list, which is sorted
by approximate distance to the destination.
```

```
  while the list is not empty
    Current path = Get the first path of the list
    and remove it
    if (the current path does not reach the
       destination) then
      if the path is not end
      then Reproduces the current path to
           new paths with its branches and
           store them back to the list
      else Ignore the path
      end if
    else Terminates the loop and the path is
         established.
    end if
  end while
```

The new list is started with the left path and right path of a stating point. Afterward, new nominated paths are constructed recursively from the extension of the existing shortest path in the list by adding its branches before the next intersection

appears and then are stored back to the list. If the new branch is leading away from the destination or the path length is increasing without leading closer to the destination then it will be pushed down from the head of the list. Repeated execution of this will gradually lead to the destination with the shortest path at the first element of the list.

From this approach, it can ensure the path is going in the correct direction and heading to the destination that will be selected. On the other hand, the limited size of the buffer will evict the last element from the list when the buffer is full. This will eliminate the unnecessary search of those branches leading in the opposite direction. In some cases, this also eliminates the infinity search when there is no path connection between two points. Figure 4 shows an example of path searching from Werribee to Queenscliff and Anglesea.

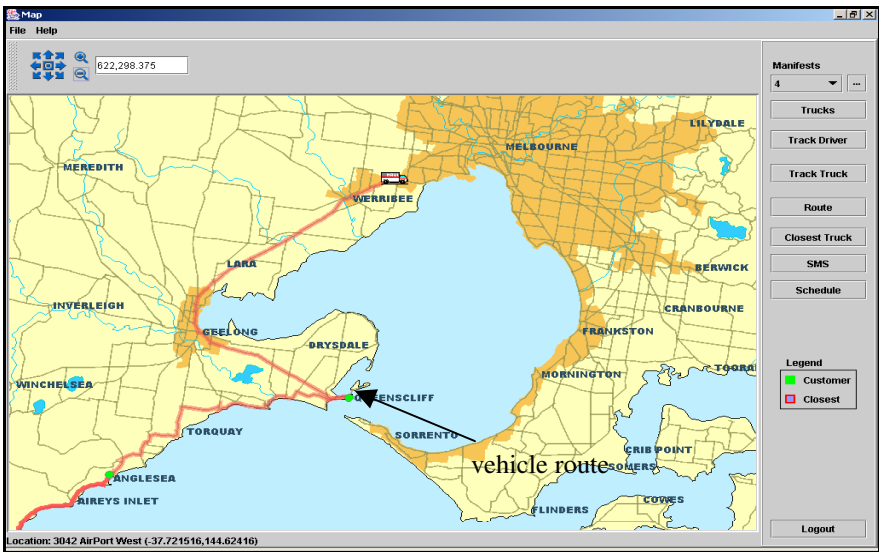


Fig. 4. Vehicle route from Werribee to Queenscliff and Anglesea.

3.2.2 Scheduling

After the routing, scheduling is a vital requirement for providing efficiency of transport or logistics services. Scheduling performs the second level of search with time and load constraints to produce an optimal route for the whole journey of vehicles. Therefore, the scheduling algorithm will be covered in this section.

Insertion Schedule

The implementation of scheduling is based on the Insertion Heuristic. The basic idea of scheduling is using the minimum number of vehicles to fulfil the pickup and delivery orders in the same day with time and load constraints. The algorithm for Insertion schedule is as follows:

```

VL: Vehicle list; OL: Order list; RL: Route list
R: Route; O: Order
while (VL  $\neq \emptyset$  )
    R  $\leftarrow$  GET-ROUTE (VL);
    while (OL  $\neq \emptyset$ )
        Order  $\leftarrow$  GET-ORDER (OL);
        for ( INSERT (O.pickup, R) )
            CHECK-TIME-LOAD-CONSTRAINTS ();
            if ( OK (Order.pickup) ) then
                do      INSERT (Order.delivery, R);
                for (START-PICKUP-till-ENDROUTE)
                    CHECK-TIME-CONSTRAINTS ();
                    If  ( OK(Order.delivery)
                    then RL  $\leftarrow$  ADD (R, RL);
                                OL  $\leftarrow$  REMOVE (O, OL);
                    end if
                end if
            end if
        end while
    end while

```

First, an initial route of a vehicle is started with a customer order from the order list. Then, another order is tried to be inserted into the initial route from the beginning position with the pickup. If the constraints are satisfied, the delivery will be insert behind the pickup with a different position for the satisfaction test. Moreover, the pickup for the new order may shift to the second position. The best result of the tests will be chosen according to the earliest time of the end time of the routes. If the vehicle is full then a next vehicle will be assigned. The process of scheduling will continue until the orders list is completed or all vehicles are full.

Time Window in Scheduling Process

MIDAS Scheduling takes the VRPTW [2] mathematical formulation to focus on the problem and illustrate the difficulty of problems with time windows. VRPTW is given by a fleet of homogeneous vehicles V and a directed graph $G = (N, E)$. The graph consists of a finite set of nodes N and a finite set of edges E . Let $N = \{0, 1, 2, \dots, n\}$, we denote the central depot as $\{0\}$ and the customers as $\{1, \dots, n\}$. The set of edges represents connections between the depot and the customers and among the customers. For each edge e has two endnotes i and j , which is denoted by $e(i, j)$, we associate a cost c_{ij} and a time t_{ij} . This cost may include service time at customer i .

Every customer in the network must be visited only once by one of the vehicles. Each vehicle has a limited capacity q , and each customer has a varying demand d_i . Each customer must also be serviced within a pre-defined time window $[a_i, b_i]$. A vehicle must arrive at the customer before b_i . It can arrive before a_i but the customer

will not be serviced before the time of b_i . The depot also has a time window $[a_0, b_0]$. Vehicles may not leave the depot before a_0 and must be back before or at time b_0 .

There are two types of decision variables in VRPTW. The decision x_{ijk} ($i, j \in N$; $k \in V$; $i \neq j$) is 1 if vehicle k travels from node i to node j , and 0 otherwise. The decision variable s_{ik} denotes the time vehicle k starts service at the customer i . The following mathematical formulae are given from [1]:

$$\text{Min } \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \quad (1)$$

subject to

$$\sum_{k \in V} \sum_{j \in N} x_{ijk} = 1 \quad \forall i \in N \quad (2)$$

$$\sum_{i \in N} d_i \sum_{j \in N} x_{ijk} \leq q \quad \forall k \in V \quad (3)$$

$$\sum_{j \in N} x_{0jk} = 1 \quad \forall k \in V \quad (4)$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \quad \forall h \in N, \forall k \in V \quad (5)$$

$$\sum_{i \in N} x_{i0k} = 1 \quad \forall k \in V \quad (6)$$

$$s_{ik} + t_{ij} - K(1 - x_{ijk}) \leq s_{jk} \quad \forall i, j \in N, \forall k \in V \quad (7)$$

$$a_i \leq s_{ik} \leq b_i \quad \forall i \in N, \forall k \in V \quad (8)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in N, \forall k \in V \quad (9)$$

The constraints (2) state that each customer is serviced exactly once, and (3) state that no vehicle is loaded with more than it's capacity allows. The next three sets of questions (4), (5) and (6) ensure that each vehicle leaves the depot, after arriving at a customer the vehicle leaves again, and finally arrives back at the depot. The inequalities (7) state that a vehicle k cannot arrive at j before $s_{ik} + t_{ij}$ if it is traveling from i to j . In formula (7) K is a large scalar. Finally constraints (8) ensures that time windows are observed, and (9) is the integral constraint.

This mathematical formulation ensures MIDAS scheduling system to achieve its objective of minimizing the vehicle fleet and the sum of travel time and waiting time to supply all customers in their required hours. These computation guarantees the fulfillment of an order in the required upper and lower time bound time frame window, which is associated with the depots.

4 Implementation and Testing

MIDAS Performance

The main performance issues of this system are the route searching and order scheduling, which are required for producing an optimal route scheduling. To find the shortest path between two locations is a time consuming process in a digital map with 470,000 connection points and 5,000 intersection points. Therefore, the search algorithm should be able to return an optimal path in 5 seconds.

In route scheduling, not only is path searching required, but also the second level of route scheduling searching is also crucial. However, the two levels of search increase the execution time. This will be not an issue for static route scheduling because it can gain extra time during midnight or off line periods. In contrast, a dynamic route scheduling requires a quick response to the customer. Thus, the

maximum decision time for accepting a same day order should not exceed more than 30 seconds.

Performance Testing

The performance testing of this section is used to measure the execution duration of static route scheduling. The sample test case is selected with four customer orders, which include eight service points (pickup and delivery), spreading over the Melbourne metropolitan area. This experiment is done on an Intel Pentium III 1GHz machine with 250MB RAM. The execution will search through the entire map of Australia with 480,000 road nodes to produce an optimal route with the time frames (Figure 5).

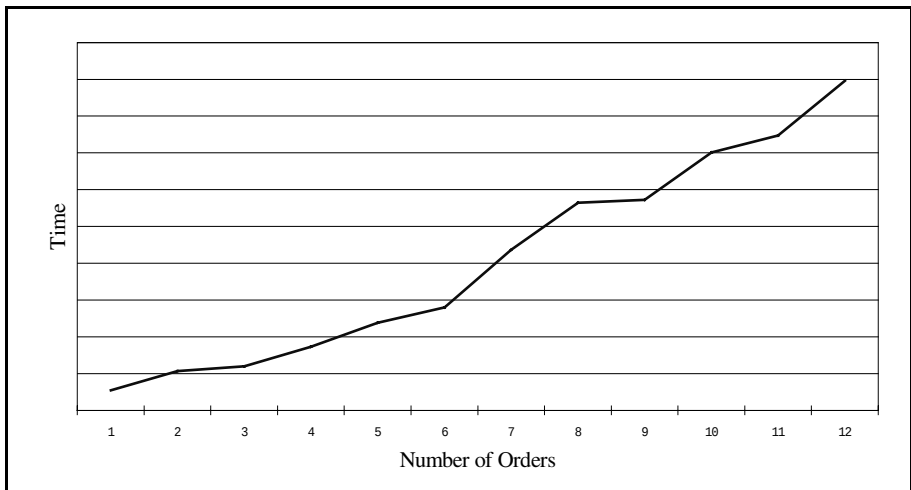


Fig. 5. The time trend against the order growth in scheduling.

5 Conclusion

The outcome of this system minimizes the ordering procedure and delivery time for effectiveness and efficiency achievement of transport and logistics services. The system had provided a comprehensive solution for the MIDAS server with digital map and SMS technologies to overcome the missing components of traditional applications. Furthermore, the route scheduling algorithm is also implemented, which include closest point nomination, path searching and insertion schedule, to perform dynamic scheduling.

The MIDAS server had offered the different functionalities for mobile device users, Internet users, vehicle drivers and system operators to interact with the autonomy system through the network connectivity. Furthermore, all these information such as ordering and tracking are stored in a backend database as conventional transportation software.

In the map component, the MIDAS server includes the Australia wide map with major roads. It has also covered the localities of major suburbs in the state of Victoria

and a few of these in other states. The map data gives the fundamental information for route scheduling. It also allows the system operator to track drivers, vehicles and routes with the digital map.

A dynamic route scheduling relies on two searches, local search and global search. Local search provides the shortest path information between two individual points. Global search accomplishes the time scheduling arrangement between multiple locations. Therefore, The map data is used to provide realistic information about the path information between two points on local search and then the Insertion algorithm is applied to perform global search on route scheduling. At this stage, the system has completely implemented the route scheduling with the assistance of the map data and Insertion algorithm to perform dynamic search.

In the near future, our aim is to extend the map software and enhance the feasibility of the scheduling functionality of the MIDAS server, such as road name and suburb name. Moreover, the street level map is essential for courier services rather than the state level. Thus, a much-advanced map is needed for future versions of the MIDAS server.

The performance of the route scheduling is also critical in terms of computation time and better solution. Besides the algorithm self-improvement, caching techniques [13][14][15] can be also considerable for redundancy work. Caching is considered useful for reducing mechanism delay. In this system, caching can be used to reduce the computation delay due to the dynamic routing in real-time, a route between two points will be computed in every execution, even if they are the same points. Hence, the application can cache the redundancy segment of the routes to prevent the recomputation delay. Especially during the scheduling arrangement, it sits on top of the path searching and requires multiple comparisons of each route at different times for producing optimal results. In addition, the cache can also reduce the rendering time of the routes on the digital map without searching for a second time.

Acknowledgment. This project is supported by the ARC (Australian Research Council) Linkage-Project no. LP0218853 and SUN Microsystems.

References

1. Larsen, J. "Vehicle Routing with Time Windows – Finding Optimal Solution Efficiently", DORSnyt, 1999.
2. Solomon, M. "Algorithms for The Vehicle Routing and Scheduling Problems with Time Window Constraints", *Operations Research*, 35(2), 1987.
3. Berger, J., Salois, M. and Begin, R. "A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows", *Proc. of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 114–127, 1998.
4. Savelsbergh, M. "Local Search in Routing Problem With Time Windows", *Annual Operations Research* 4, 285–305, 1985.
5. BBN Technologies <http://openmap.bbn.com>
6. Environment System Research Institute, Inc. <http://www.esri.com>
7. Environment System Research Institute, Inc. "ESRI Shapefile Technical Description", An ESRI White Paper, July 1998.
8. Hoch, F. "Assessing a wireless future", *Trends Report* 2001, Oct 1, 2001.
9. BlueSkyFrog wireless service provider. <http://business.blueskyfrog.com>

10. Geoscience Australia <http://www.ga.gov.au>
11. Telstra Mobile SMS Access Manager. Technical Guide.
http://www.telstra.com.au/mobilenet/pdf/sms_techguide.pdf
12. Solomon, M. and Desrosiers J. "Time Windows Constrained Routing and Scheduling Problems", *Transportation Science*, 22(1), 1988.
13. Castro, M., Adya, A., Liskov B., and Myers, A.C. "HAC: Hybrid Adaptive Caching for Distributed Storage Systems", *Proc. of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, 1997.
14. O'Neil, E. J., O'Neil, P. E., and Weikum, G. "The LRU-K Page Replacement Algorithm for Database Disk Buffering", *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp.297–306, 1993.
15. Robinson, J., and Devarakonda, N. "Data Cache Management using Frequency-based Replacement", *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 134–142, 1990.
16. Malhotra M and Tari T. "MIDAS – An Integrated E-Commerce Solution for Australian Transport Industries", *International Journal on Web Engineering and Technology (IJWET)*, 2003. To appear.

An Approach to the Process Maps Based on Semantic Web Methodologies

Roberto Fresco¹ and Andrea Pederiva²

¹ Information Science and Technologies Institute (ISTI),
CNR Research Area

56124 Via G. Moruzzi 1, Pisa, Italy
Roberto.Fresco@isti.cnr.it

² Deloitte Consulting,
31100 Piazza S. Vito 37, Treviso, Italy
APederiva@dt.deloitte.it

Abstract. Organizations use process mapping as a basic tool to define or analyze how they manage their business. Process mapping methodologies are also used to design better business processes, to improve efficiency and foster productivity and output quality, while reducing business risks. Process maps are usually developed by business analysts and process engineers, which are often highly centralised knowledge resources and generally apply process mapping methodologies based on top down approaches. Business analysts are used to represent process maps using flowcharting tools, or, in case of database supported knowledge bases, using tree-shaped structures. Unfortunately, top-down approaches and both flowchart-based or tree-shaped process knowledge bases lead to process maps which are mainly unable to represent in a complete manner the enterprise knowledge about its processes and generally hinder the sharing and the subsequent reuse of the process maps. In this article we illustrate an alternative approach to represent process maps and business knowledge, based on typical concepts and methodologies from the Semantic Web paradigm (i.e. ontologies, XML based languages and agents), and aimed to allow for the production of complete, meaningful and reusable process maps.

1 Introduction

By common definition, a Business Process (BP) consists of a group of activities (time ordered) or steps and their relationships, aimed to realize defined organization objectives. We can consider a process as a “value chain” by its contribution to the creation or delivery of a product or service to the benefit of the enterprise and its customers. BP will include several general services in the process, and some of those services depend upon the business’s enterprise network. One may think of a service as an abstraction over and above the enterprise network.

Each step should add “value”, where the value of an activity has the meaning of those process results valuable to the customers. In other words an activity that adds value is

one that is requested by the customer and that the customer is willing to pay for. Value added activities should be optimized under a value added/resources perspective, whilst non-value added activities should be optimized and justified on a cost/risk mitigation perspective. For this purposes enterprises need to improve or re-engineer business processes [1] in order to eliminate not necessary activities, simplify processes and better combine related activities.

The classical approach to process modeling and mapping consider a top down strategy; it starts from strategic objectives, usually designed by top management and it subsequently describes in more detail the procedures of the business units of the organization.

In this view organizations are managed and measured vertically, but in fact work gets done horizontally via Business Processes. We must consider that organizations change over the time, causing processes to become fragmented or even ineffective and forcing the management to take into account increasing organizational risks.

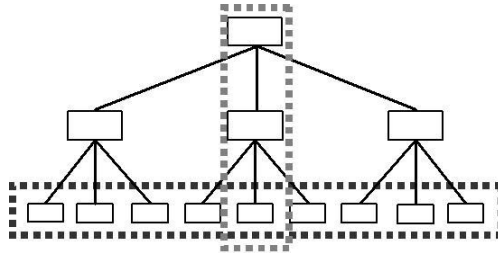


Fig. 1. Processes cut across functions and levels of organizations. A horizontal structure is needed to avoid risk and achieve business goals.

Today, processes cut across functions and layers of the organization to fit the needs raised by the instant access revolution (i.e. via WWW) which contributes to broke down geographical and economical barriers and lead to full and open competition. Here is the need to share knowledge about processes across different geographical sites and functions, in order to support organizational collaboration. So we consider that top down process mapping methodologies are inadequate as they don't enforce as much collaborative and distributed effort on process modeling and mapping activities as would be possible, and useful.

2 Process Mapping

Generally, a process has inputs and outputs and a particular output can be the input for another process (i.e. there are process interfaces). Processes are generally represented at different levels of detail, since a single process can be decomposed into sub-processes or sub-activities. The outputs can be semi-finished products or deliverables, when the produced product, service or information is in a form that allow for easy use

in other processes. In fact it is reasonable to talk about business activities, business documents, business objects, business events, business rules, roles and resources. According to IDEFØ standard a process can be represented as in Fig. 2. IDEFØ assists the process modeler in identifying the functions performed and what is needed to perform them. An IDEFØ model diagram is then composed of several activity boxes and relationships to capture the overall activity.

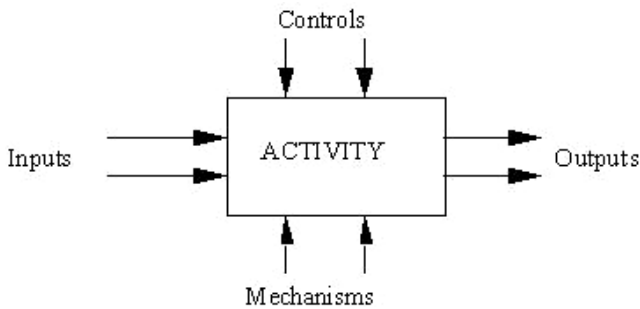


Fig. 2. IDEFØ syntax: Activities process Inputs into Outputs using Mechanisms according to the processing rules described in the Controls

Other IDEF formats were defined to describe the generic process modeling phases [2]. Process mapping can be thought of as a set of graphical methods with the emphasis being on the working processes carried out by people, and on the broader flows of information across an organization.

Examples of process maps are flowcharts or oriented graphs and in [3] a formalism based on the graph grammars was proposed. It is a common practice in the enterprises world the storing of knowledge about processes, in hierarchical form (i.e. a tree view), as it is more compact and convenient to store in databases and spreadsheet system.

Unfortunately this approach has a large number of drawbacks and the most important one is related to the fact that process mapping becomes a *one shot* effort: the process maps are useful within a short time following their production.

A reason of this inconvenience is related to the use of such data structure (the tree) that is not useful in representing the overall process Knowledge. The Knowledge of the expert who created the process map is implicit and not well explained by the tree view. Without their owner guidelines, process maps in this hierarchical view can lead to misunderstandings and gaps in overall activities.

There is also a loss of meaning and learned skills when tree maps pass from its own creator to another people within the same organization.

Organizations can prevent the impairment of their process knowledge bases representing enterprise processes using oriented multi-level graphs that shows a better vision of the knowledge of the process and allow for the contribution of de-centralized skills to the documentation of the organization business practices.

The main rules one has to follow to succeed in using multi-level graphs in mapping processes are the following:

- every activity has inputs and output,
- the outputs of an activity can be the inputs of another activity,
- when an activity is detailed with a graph at the subsequent level, the inputs of the activities in the graph at the subsequent level must be the inputs of the exploded activity; the same holds for the outputs.

Following these rules one can document process knowledge also with a bottom-up approach; with minimal process mapping skills, and without an overall knowledge of the entire business operating, people can document the business activities they are directly involved into; then, by linking activities inputs and outputs and grouping activities at higher levels of the process map structure, organization complete process maps can be generated.

Considering that every activity has inputs and outputs, we can easily use many based hypertext and hyperlink navigation tools [4] that store organization procedures and have automatic functions to pass from graphical view to structured data view and vice versa. These tools can be used in Intranet environments with the aim to diffuse the enterprise knowledge to its staff.

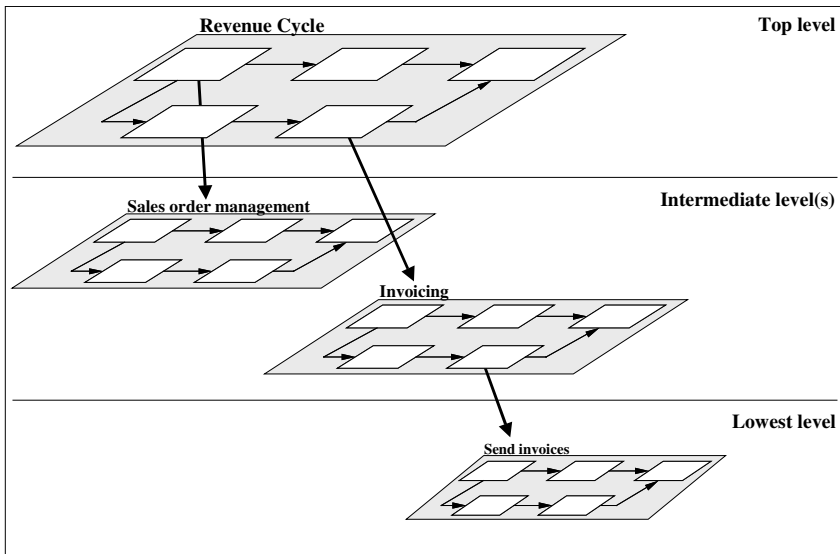


Fig. 3. An example showing the multi-level graph paradigm used to chart the revenue cycle processes.

While the multi-level graph process mapping approach is already available (but not widely used), the lack of a precise syntax and semantic to represent business processes still prevents this approach from being able to maximize the value it can bring to businesses, and allow for people to continue to make common mistakes, such as using too much simplified representations or still exporting process knowledge using tree-shaped formats. A definite meta-model to express process-related functionalities

should also support the modeling and analysis of business risks related to process activities.

It is right to distinguish processes from *procedures* that are by definition a detailed subset of a process that receives inputs from it, and carries out a logical set of steps, returning a result as part of the overall ‘master’ process. (Procedures describe how you carry out what you do as part of a process). These concepts can be considered as the meta-model for a generic process and can help to show how processes/procedures link together (connectivity) within the particular process model created for a specific purpose.

3 A Semantic Web Approach for Business Processes

As we have seen, process mapping can usefully begin at the *bottom*, i.e. from the mapping of the more detailed activities, rather than from the top, i.e. from the listing of the main business processes. The idea behind process mapping is to define which is the goal of the process, and then to illustrate how that goal is obtained.

The bottom-up approach to process mapping makes the most of the contribution from the organization human resources. Key personnel should be identified, and contacted to gain co-operation. The process mapping efforts of key personnel usually supported (but not substituted) by experienced internal or external consultants will provide complete and detailed process maps at the operational level. Appropriate collection of process attributes will provide useful information to be exploited to find innovative, effective practices.

The process mapping effort at the operational level should be performed process by process in a horizontal way, considering also the process context. All this contribute to extrapolate the relevant information. Once the information had been collected, analyzed, structured, the next work would be the realization of conceptual models, that we can name enterprise *ontologies* [5].

Ontologies represent the key concept of the Semantic Web and they are specific artifacts designed with the purpose of expressing the intended meaning of a shared vocabulary [6]. Ontologies are formal specifications of terms and relationships among them; in other words ontologies *constrain* the intended meaning (i.e. an agreement) as they are a specification of conceptualization. The conceptualization is a formal structure of a piece of reality as perceived by an agent (human or software actors involved in the organization) independently of the terms used and actual situation. In fact different situations involving the same object but described by different vocabularies, (i.e. *fattura* as the Italian term for invoice), may share the same conceptualization. Using ontologies and the “bottom-up” approach as explained above, synonyms and homonyms can be defined in order to eliminate the cross-functional terminology gaps experienced by people belonging to different departments.

In fact the issue is how to allow for different people (domain experts and not) to easily share knowledge and how to give them the capability to understand each other by “speaking” the same language. Moving towards this objective one discover the need

for methods to describe the semantic of business process maps in a machine-understandable fashion. Once business process semantics are machine-understandable, we can think the cooperation among different enterprises (i.e. virtual organizations) to become possible. We can obtain the conversion of information from one ontology to another, so that the results of queries and operations on resources described by different ontologies can be combined. Moreover we can easily think about the presence of a large number of software agents specialized in using their own ontology and cooperating among them to produce relevant process information [7].

We will discuss in the next paragraph a possible implementation of the discussed concepts, based on the revenue cycle example introduced above.

Let us make the following assumptions:

- we have a manufacturing firm whose Revenue Cycle consists of the following main sub-processes: Grant Credit, Collect Sales Orders, Ship Products to Customers, Invoice Customers, Manage Cash Collection, Record Sales and Accounts Receivable;
- the invoices are prepared by a Sales Management Department, while they are sent to the Customer by a corresponding secretary;
- the inputs to invoice preparation are "Shipped orders" and "Customer sales and payment conditions", while the output is "Prepared invoice"; the input to send the invoice is "Prepared invoice", while the output is "Sent invoice".

Now, the "Prepare invoice" process can be documented by a Sales Management Department clerk, while the "Send invoice" process can be documented by the corresponding secretary, which are the people that better know those processes. All we have to do to have the map of the "Invoice Customers" process is to recognize that the output of the "Prepare invoice" process is the input of the "Send invoice" process. Recognizing this allow us to have a graph of the "Invoice Customers" process at the upper level with two activities, "Prepare invoice" and "Send invoice", connected by the output/input item "Prepared invoice".

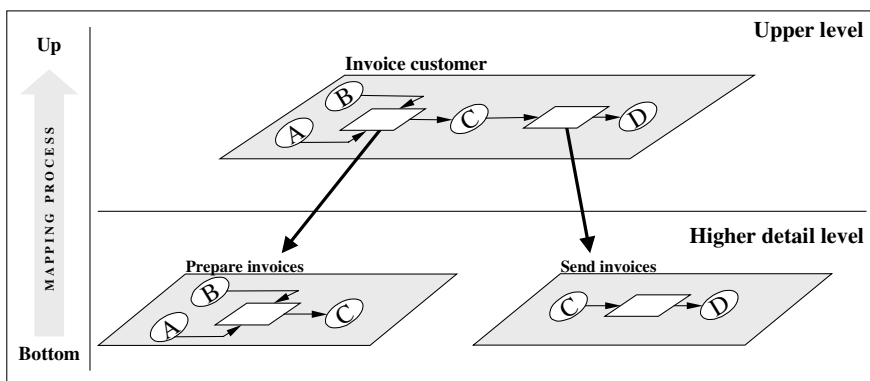


Fig. 4. An example of the bottom-up process mapping approach based on multi-level graphs. A and B are respectively "Shipped orders" and "Customer sales and payment conditions" inputs; C is the "Prepared invoice"; D is the "Sent invoice" output.

Now, with the same approach we can map the "Ship Products to Customers" process. Recognizing that this process has a "Shipped order" output, we can go further, and have at the upper level (a piece of) the Revenue Cycle map.

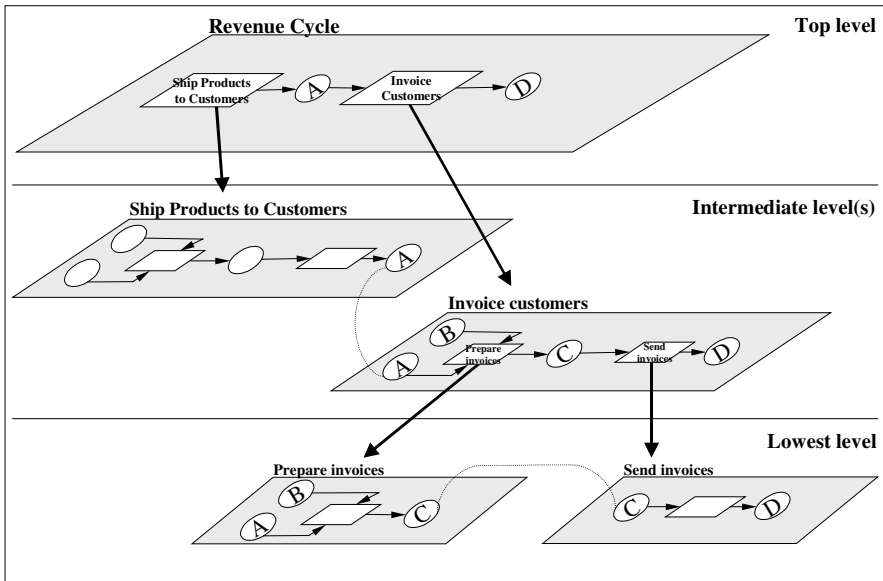


Fig. 5. Adding the "Ship Products to Customers" process.

A fundamental issue is to have the organization knowledge about processes represented graphically in the multi-level graph shaped charts and recorded in appropriate knowledge bases fully aligned. This can and must be accomplished using appropriate tools and methodologies enforcing the use of a precise syntax and semantics both at the graphical and database level. Most of the not strictly graphical attributes can be recorded only in the process knowledge databases and shown in the charts only in specific purpose customized reports.

What is important here, is to recognize that if we have a common language to identify inputs and outputs the work of analyzing the process maps at the lowest levels (the deeper detail levels) can be done automatically, by analyzers, or by co-operating agents which only have to share a common language (ontologies) at the interface level.

If business processes follow an Ontology-based design, definitions are project-neutral and can thus be used in a cross-project fashion. Furthermore, dependency on tool vendors is reduced, since ontologies are increasingly based on public standards (e.g. RDF, OWL). In this way, Business analysts are free to define their own metamodels.

It is important to build an overall knowledge base that can be useful in the typical phases of organizations life. Following this approach, the entire semantically process definition is kept in a single place. In this way there is no need to keep process defini-

tion elements scattered around over multiple repositories, created and edited with various costly tools.

Business process definitions are stored in XML, which makes it fairly easy and straightforward to generate process definitions using different syntax (e.g. UML activity diagrams, Event-driven Process Chains etc.).

In fact it is possible handle UML diagrams, that are already in organizations, for obtaining XML syntax by XMI specification and RDF schemas using the XMI to RDFS stylesheet [8].

3.1 Possible Use of RDF in a Business Process Context

In the Semantic Web Stack, shown in Fig.6, RDF represents an XML-serializable language for expressing metadata for resources on the Web [9].

XML's "tag-centric" syntax makes it a natural fit for representing ordered sequences and hierarchies. Thus it is well suited for ordering time points and occurrences of activities. It is also good at representing sub-activities and sub-occurrences.

The RDF data model is essentially a labeled directed graph, and an arc or node label may either be a URI or a literal.

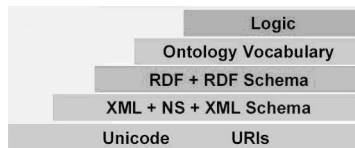


Fig. 6. The Semantic Web Stack as revealed by Tim Berners Lee.

Although RDF is designed for specifying metadata about resources on the Web, a basic type system defined for RDF enables RDF to be used for specifying schemas. RDF is probably the best choice for representing the objects used in a process. RDF can be embedded into process descriptions as needed, and RDF Schema [10] is useful for specifying object structure, classes and instances, and inheritance relationships. As a matter of fact, RDF/RDF Schema could be used to represent not just the objects, but also the entire process description. RDF is a framework to describe resources and their properties. The basic concept is a triple, describing a resource, a property of that resource and its property value. The property value can be another resource or a literal value. There is a RDF/XML syntax that expresses the graph notation and there is the need to have common namespaces to identify resources in our schema.

In general it is possible to express activities and sub-activities and hierarchies. So we can establish links between activities (i.e. one is the input of the other).

We illustrate an example of RDF/XML related to the process "Choosing a PC supplier". We have classes Company (Supplier and Customer are subclasses of Company), Product. The property that a particular company supplies a product is indicated as "Sells":

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:ID="Company">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Supplier">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Company"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Customer">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Company"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Product">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
  </rdf:Description>
</rdf:RDF>

```

To refer to these classes in RDF instance data (e.g., data describing suppliers) located elsewhere, we would need to use the full URIs (namespaces) to identify the classes.

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/schemas/Company">
  xmlns:exsup="http://example.org/schemas/suppliers">
  xmlns:exprod="http://example.org/schemas/products">
  <rdf:Description rdf:ID="supplierPC">
    <rdf:type rdf:resource="http://mysupplier.org/sup.rdf#CDC"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Notebook">
    <rdf:type
rdf:resource="http://myproduct.org/prod.rdf#NotebookAcer"/>
  </rdf:Description>
</rdf:RDF>

```

We describe property about a supplier sells a product

```

<rdf:Description rdf:ID="Sells">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
  <rdfs:domain rdf:resource="#Supplier"/>
  <rdfs:range rdf:resource="#Product"/>
</rdf:Description>

```

As we described in the Revenue Cycle, the activity “prepare invoice” consists of two inputs Shipped Orders and the “Customer sales and payment conditions”. And the object Invoice is declared as a class in the same way as above for the class Customer etc. We suppose to have the “Customer sales and payment conditions” data in a file `paym.rdf` and the shipped orders in a `sh_order.rdf`

```

<activity id="a4">
  <name>prepare invoice</name>
  <activity id="a5">
    <name>shipped orders</name>
    <resource rdf:resource=" sh_order.rdf#ID123 "/>
  </activity>
</activity>

```



```
</activity>
<activity id="a6">
  <name> Customer sales and payment conditions</name>
  <resource rdf:resource=" paym.rdf#IDAb1623 "/>
</activity>
</activity>
```

4 Applications

Bottom-up process mapping approaches based on multi-level graphs and supported by a precise syntax and semantic can boost the efficiency of the process mapping efforts, especially by making the most of the operating people process knowledge, and reducing the effort of expensive, centralized resources, such experienced people or consultants.

Moreover, the involvement of the operational people allow for process maps to become a real organization knowledge asset, enabling effective knowledge sharing. Indeed, to become effective as a knowledge sharing tool, bottom-up process maps based on multi-level graphs must be supported by appropriate navigation tools, usually based on intranet tools exploiting the hyper-text capabilities of the internet browsers.

The involvement of the operation people and the diffusion of the process maps throughout the organization is fundamental to achieve another essential (and usually failed) objective: to maintain up to date the process maps. If we can assume that we have the process maps realized by operational people at the level of detail they know, and the process maps at the upper levels built by analyzers or automatic agents, we can imagine a de-centralized maintenance process for centralized organization process knowledge bases.

Moreover, the alignment of graphical and database supported knowledge bases on the organization processes open many interesting scenarios on the re-use of process maps in different context.

Generally, when organizations begin an organizational change or improvement effort, such as "quality", "process efficiency", "internal controls", "auditing", "security" initiatives, specific, purpose-oriented one-shot process maps are realized. With this approach, the organization knowledge on its processes become fragmented and usually impossible to update.

If we can imagine that we have a centralized knowledge base on the organization processes, with a decentralized maintenance process, associated with appropriate capabilities to align graphical and database process representations, we can imagine to have the database representation of processes enriched with the appropriate attributes needed to manage the "quality", "internal controls", "security", and maybe "law compliance" objectives of the organization.

Indeed, a company organizational structure and process do not change if we look at them with the glasses of the "quality manager", the "auditor", the "security officer" or the "regulatory authority inspector". What change is which processes we are mainly interested in, and which process attributes we want to manage. But this can exactly be done having appropriate "views" on a structured process knowledge base. And the

ontology design takes into account the context in which end users create and consume information. It is important to build a new generation of intranets in organization where there are continuous needs of exchange of information and news to improve business processes and a framework based on ontologies and XML serializable languages (i.e. RDF) helps to build a common language of the enterprises.

5 Conclusion

The introduction of semantic in Business process mapping is an open challenge: We examined some problems and solutions based on Semantic Web approach and the ontology and RDF code illustrated as example are intentionally simplified. The conclusion is that such visions could be realized, but require a consistent approach involving all the process strategies and there is the need of developing tools for the market. Only time will tell.

References

1. Mayer, R.J., deWitte, P.S.: Delivering Results: Evolving BPR from art to Engineering (1998)
2. <http://www.idef.com>
3. Klauck, C., Müller H.J.: Formal Business Process Engineering based on Graph Grammars (1996)
4. <http://www.triaster.co.uk>
5. Fox, M. S., Grüninger, M.: Ontologies for Enterprise Integration, (1994)
6. Guarino, N.: The role of ontologies for the Semantic Web, Semantic Web Tour, Rome (2003) <http://www.w3c.it/events/st/guarino.pdf>
7. Fensel, D., Bussler, C. et al.: Semantic Web Application Areas (2002)
8. Cranefield, S.,: Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital Information Vol.1 Issue 8 (2001)
9. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, 22 February 1999, <http://www.w3.org/TR/REC-rdf-syntax>
10. Resource Description Framework (RDF) Schema Specification, W3C Proposed Recommendation, 3 March 1999 <http://www.w3.org/TR/PR-rdf-schema/>

PEPt – A Minimal RPC Architecture

Harold Carr

Sun Microsystems, 4140 Network Circle, SCA14,
Santa Clara, CA 95054 U.S.A.
harold.carr@sun.com

Abstract. We present a high-level RPC architecture. Although RPC systems seem quite varied they actually share the same fundamental building blocks. We examine the operation of several technologies (e.g., SOAP/HTTP, RMI-IIOP) to show what they have in common, namely four main blocks that we call PEPt: Presentation, Encoding, Protocol and transport. Presentation encompasses the data types and APIs available to a programmer. Encoding describes the representation of those data types on the wire. Protocol frames the encoded data to denote the intent of the message. Transport moves the encoding + protocol from one location to another. The PEPt architecture allows one to understand, use and implement RPC-systems by providing a simple but comprehensive framework in which to place finer-grained details. It may also serve as the basis for RPC infrastructure reuse between seemingly disparate systems. The PEPt architecture *enables an RPC system to adaptively change encodings, protocols and transports.*

1 Introduction

The specification and implementation of Remote Procedure Call (RPC) [1] systems such as DCE [2], distributed versions of C++ [3][4], COM/DCOM [5], CORBA [6], RMI [7], RMI-IIOP [8], XML-RPC [9] and SOAP [10], seems to traverse the same ground repeatedly. One way to avoid reinventing the wheel is to isolate the basic building blocks. This may seem difficult since, at first glance, it may seem RPC systems have nothing in common. We claim that these systems are variations of a fundamental architecture we call PEPt.

We show the PEPt (Protocol, Encoding, Presentation, transport) architecture as a high-level way to structure our thinking, design and implementation of RPC systems. The PEPt architecture has been used in a commercial CORBA system [11]. PEPt embodies the experience of evolving this system from C++ to Java and responding to its changing requirements over time (e.g., alternate protocols and transports, as well as revisions in stubs and encodings). If you build or use more than one RPC system, then the PEPt architecture will help you organize your approach to RPC by providing a clear structure that: provides clarity as to where a function belongs, makes it easier to evolve the system over time, is comprised of a small number of pieces that is easy to

hold in one's head, and specifies a simple decomposition of RPC systems which universally applies.

2 Overview

How can we become better at designing, specifying, building and maintaining RPC systems? We can do so by defining an architecture that is simple enough to hold in the mind as a whole, while being comprehensive enough to describe and implement diverse RPC systems. We show how the PEPt architecture supports the client-side operation of stubs and the server-side operation of ties. We show that the common structure is symmetric: on the client-side a programmer makes a remote call with arguments of specific types (presentation). The types are converted into a representation agreed upon by both the client and server sides (encoding). The encoding is framed with information that carries the intent of the message (protocol). The raw bits of the encoding + presentation are moved from the client location to the server location (transport). The server side goes through these steps in reverse until it obtains the arguments to call the procedure. The whole process repeats itself to return a result. All RPC systems either implicitly or explicitly carry out these steps. PEPt gives us the ability to structure our thinking about RPC systems in such a way as to allow us to build scalable, reusable, maintainable infrastructure.

3 Related Work

The ADAPTIVE Communication Environment (ACE) [12] represents seminal work on applying patterns to network programming. Whereas ACE is a complex system more specific to C++ and high-performance (and, to a lesser extent, CORBA) PEPt is a higher-level, language-independent view of RPC not tied to a particular type of RPC system. PEPt presents an architecture for RPC with fewer "moving parts" in order to guide the overall structuring of a system. ACE focuses more on the details whereas PEPt focuses on the big picture. They complement each other.

RM-ODP's [13] engineering viewpoint channel model is similar to PEPt but does not define important interactions such as how two binding objects interact and connection multiplexing. Although ACE and RM-ODP provide useful patterns they do not give enough detail regarding the overall interaction between the top-level parts of the system.

The SOAP and WSDL [14] specifications allow one to specify different transports and encodings. PEPt is an architecture in which to implement such specifications.

The Jini extensible remote invocation portion of the Jini Davis project [15] is an API and architecture to enable alternate transports and encodings. It is focused on Java's RMI programming model whereas PEPt is language and programming model independent.

Subcontracts [16] were proposed as a way to define new object communication without modifying the base system. PEPt is similar but limits subcontracts to the protocol block, and further delineates that block's relationship to other subsystems to enable finer-grained control and reuse.

PEPt is an architecture for RPC in the same way that others have created architectures and frameworks at different levels of the network such as SASL [17] for security, BEEP [18] for application protocol and Boecking's research [19] in network protocols.

4 Fundamental Building Blocks

The fundamental building blocks of RPC systems are: *Presentation*, *Encoding*, *Protocol* and *Transport*. This paper refers to these blocks as a group as PEPt. Fig. 1 shows PEPt's core architecture.

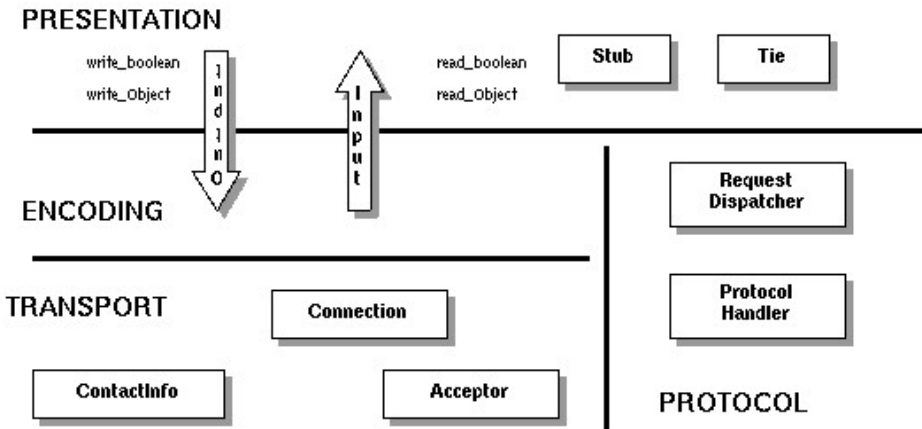


Fig. 1. PEPt architecture with primary interfaces

The boxes in Fig. 1 (e.g., ContactInfo, Input) represent interfaces that represent or bridge the blocks of the core architecture. (Note, the blocks are purposely drawn to *not* suggest a layered architecture.) Each PEPt block is responsible for a key RPC operation.

The **presentation** block includes the APIs used to interact with an RPC system (e.g., stubs/ties), the data types that may be transferred, and error reporting.

We use the term "**encoding** block" to denote the "wire" representation of presentation data types and the conversion process from language representation to wire representation.

Data by itself makes no sense. One needs to include other information along with the data to indicate the intent of the data. The **protocol** block is responsible for "framing" the encoded data to indicate the intent of the message. On the sending side, the protocol block frames the encoded data with the intent of the message. On the receiving side it interprets the intent. The protocol block says and interprets what a message means.

The **transport** block moves a request or response (i.e., the encoded data and protocol framing) from one location to another. The most common transport today is TCP/IP. CORBA IIOP requests and responses use TCP/IP as their transport. SOAP often uses HTTP as a "transport". However HTTP is a protocol in its own right which uses TCP/IP as its transport. Besides carrying the basic SOAP message (encoding + protocol) HTTP needs its own protocol bits. In general, PEPt views the transport block as a source or sink from which you receive or send bits with no further need for PEPt to deal with additional protocol information. In that case it is clear that CORBA IIOP is a protocol and TCP/IP is a transport. In the SOAP/HTTP case, PEPt would view HTTP as a protocol, framing the SOAP message that, in turn, frames the encoded data. The entire HTTP protocol plus SOAP payload is then given to a TCP/IP transport. PEPt is flexible enough to allow various degrees of coupling between the transport and protocol blocks to handle multiple layers of protocols, as in the SOAP/HTTP case. Once the protocol block is done forming a message it gives it to the transport block to send. Conversely, when the transport block receives a message it gives it to the protocol block for handling. The transport block is responsible for transferring requests and responses from one location to another.

A question naturally arises: why these blocks? Why not more, less or just different blocks? If we look at the related work cited above we can see the subcontract-based architecture in a sense has one block, the subcontract itself. A subcontract is responsible for all variations in protocol, encoding, etc. While a subcontract is a useful pluggability mechanism it does not provide enough structure to help organize the parts that can vary. In other words, subcontracts are too coarse-grained. The ACE architecture goes the other direction: providing multiple "blocks" for network programming. However, ACE's multiplicity is difficult to communicate and easily hold as a whole. We have found, through experience, that PEPt's four main building blocks are a useful division of concerns to answer placement of more detailed functionality and to understand as a whole.

The PEPt architecture is based on our experience with other RPC architectures that tried to completely decouple architectural blocks except for a few well-known interactions. However, when the need to support features such as GIOP fragmentation or SOAP parameters encoded as MIME attachments arose, it was noted that there needs to be a closer coupling between the transport, encoding and protocol blocks. PEPt enables well-known interactions between blocks but also allows private contracts between blocks. For example, if a protocol supports fragmentation, then the encoding block will need to signal the protocol block when the encoding block's internal buffers are full, even though marshaling may not be complete. The protocol block will need to form a fragment message and give it to the transport block to be sent. The PEPt architecture allows such coupling in a generic manner.

Now that we have introduced PEPt's fundamental blocks we continue by showing them processing requests and responses.

5 Request/Response Lifecycle

By following a request all the way through, in detail, on both the client-side and the server-side, it can be shown that PEPt's fundamental blocks provide the right level of granularity to implement RPC systems. We will list the steps necessary to support stub operation and we will show how the PEPt architecture supports those steps.

5.1 Client-Side Lifecycle

The steps to support a remote call are: 1: Get a connection to the service. 2: Get an output stream for the connection. 3: Marshal the arguments into the output stream. 4: Send the arguments to the service. 5: Wait for a response. 6: Get an input stream for the connection. 7: Unmarshal the return value or exception from the input stream. 8: Return normal result or throw exception result. 9: Release any resources used in the remote call. The remainder of this section shows how PEPt supports these steps.

Obtaining a Remote Reference. We do not discuss obtaining remote references in detail here. The main point, in terms of PEPt, is that obtaining a reference generally results in a stub being created in the client. The stub contains the service's address information and code that (un)marshals data from/to the service. (The address information may contain alternate addresses and other information such as transactional and security requirements.) Once we have a stub we can invoke remote procedures (methods).

Invoking a Remote Reference. When a client calls a remote service the client is actually making a call on a stub. A stub is responsible for interfacing with the RPC infrastructure to accomplish the remote call. A stub is part of PEPt's presentation block: the programming model and data types applicable to that model.

Obtaining a Connection to the Service. The stub interacts with the PEPt architecture to service the request. The first step taken is to obtain a connection to the service in order to transport request and replies.

To obtain a connection it is necessary to determine the type of connection and have a factory for the chosen type. To accomplish this the client-side of the PEPt transport block has two main interfaces: `ContactInfo` and `Connection`. `ContactInfo` is an abstract representation of remote references and a factory for `Connections`. `Connection` is the interface used to transport requests and replies.

The stub interacts with the protocol block which interacts with `ContactInfo` to determine information such as location, transport, protocols, encodings, transaction,

security, and to create a specific type of Connection. The protocol block interacts with the Connection by sending and getting raw bits transported by the Connection. (We note that Connection and ContactInfo, along with the Acceptor discussed below, are a form of the Acceptor-Connector design pattern [20].)

Since a Connection may come in many forms: shared memory, Solaris Doors [21], a TCP/IP Connection, ATM, etc., other blocks in the system should not know the specific type of transport being used. In particular, the presentation block should not know anything about the type of Connection. In fact, the type of the Connection (transport), the encoding and the protocol should be able to change dynamically between invocations with no changes necessary at the presentation block. For example, it may be useful to use SOAP/HTTP when an RPC needs to traverse the Internet, but, within an enterprise, using an encoding, protocol and transport that utilizes the internal reliable LAN may be more appropriate.

To obtain a Connection the protocol block interacts with ContactInfo (this protocol block interaction is discussed later). For CORBA this may mean examining an IOR that may contain a TCP/IP host/port pair. Since the CORBA IIOP protocol allows request/reply multiplexing on single connection, an existing Connection may be used or a new Connection may be created if one is not found.

The point at which a Connection is obtained is dependent on the features supported by a specific type of RPC. In RMI-IIOP, Connections are obtained before marshaling because of GIOP fragmentation and Portable Interceptors. (If a GIOP implementation supports fragmentation and if a Portable Interceptor adds service contexts to the GIOP header which overflow the internal buffer containing the encoded header then one or more fragments may be sent. One needs a Connection in order to send a fragment. Thus the Connection must be obtained before marshaling.)

A PEPt implementation of RMI-IIOP would interact with ContactInfo to determine and create the appropriate Connection. In this case, ContactInfo would abstract an IOR. The IOR may contain multiple profiles or tagged components that specify different ways to connect to the service.

PEPt uses the ContactInfo and Connection interfaces of the transport block to enable alternate transports. We will see later how ContactInfo also serves as a factory to enable alternate encodings and protocols. Thus, ContactInfo is the primary client-side pluggability point in the PEPt architecture.

Once we have a Connection to a remote service we need a way to write and read data on the connection. That is discussed next.

Get an Output Object for the Connection. The purpose of a transport block Connection is to carry requests and responses between peers. The actual forming and processing of those requests/responses takes place in other PEPt blocks. To form the request the procedure arguments must be encoded. In other words, there must be a way to convert from the presentation block representation of arguments to the RPC representation (encoding) of those arguments. In PEPt, OutputObject and InputObject are encoding block interfaces that contain and hide the encoding from other blocks. We will discuss how they are obtained and used next.

Once a transport Connection is obtained it is necessary to obtain an `OutputObject` to be used for marshaling data. One could ask the Connection for an `OutputObject`, but that would limit the `OutputObject` to one type of protocol association and it would limit the Connection to one encoding/protocol combination. Since the remote reference (which is represented in PEPt by `ContactInfo`) contains the necessary information on what encodings and protocols may be used, it serves as a factory for the `OutputObject`.

An `OutputObject` serves several functions. Its interface defines the presentation block data types that may be written to the `OutputObject`. Its implementation defines the encoding of those types. Its implementation also defines a private contract between the `OutputObject` and the Connection on how that encoding is stored before being sent (e.g., as an array of bytes).

Once the `OutputObject` is obtained we can marshal presentation block data into it, which we discuss next.

Marshal the Arguments into the OutputObject. At this level, marshaling is simple. The presentation block stub gives presentation block data types to the encoding block `OutputObject` to encode and temporarily store in internal storage.

In RMI-IIOP marshaling is actually quite complicated since it must support chunking, fragmentation, indirections, etc. Likewise, SOAP marshaling can become involved in order to support MIME attachments. For example, to support a feature such as GIOP fragmentation PEPt allows encoding block `OutputObjects` to make private contracts with the protocol block and with the transport block Connection. These contracts enable encoded buffers in the `OutputObject` to be sent on the Connection before the presentation block is done marshaling.

Marshaling Complete, Send Arguments to Service. After it has finished marshaling arguments, the stub signals the PEPt architecture that request argument marshaling is complete. At this point the encoded arguments (or the last fragment of encoded arguments) need to be sent over the transport. Before the encoded data is actually sent by the PEPt RPC infrastructure it must be framed by protocol information. Protocol framing is the responsibility of the protocol block `RequestDispatcher` interface. `RequestDispatcher` is responsible for managing necessary headers (and trailers if present), and for giving the `OutputObject`'s internal encoded data buffers to transport to be sent on the wire.

How do we obtain an appropriate `RequestDispatcher`? Since `ContactInfo` abstracts the encoding/protocol/transport combinations available for a specific service it serves as a factory for protocol block objects (as well as transport and encoding block objects).

There is a bootstrap issue here that we will only touch upon lightly. Since the protocol block coordinates interactions between the other blocks, what interface is responsible for initially interacting with `ContactInfo` in order to choose and create a `RequestDispatcher`? PEPt handles this by associating a generic `RequestDispatcher` with the stub. The generic `RequestDispatcher`'s function is to interact with `ContactInfo`

to choose and create a specific `RequestDispatcher`. Then the specific `RequestDispatcher` takes over. The specific `RequestDispatcher` then interacts with `ContactInfo` to create the `Connection` and `OutputObject`.

Generally the choosing and creation of `RequestDispatcher`, `Connection` and `OutputObject` will occur when the stub obtains an `OutputObject` for marshaling. This is usually the case since protocol information may need to be marshaled into the `OutputObject`'s internal encoded data storage even before beginning argument marshaling. There are two primary examples of the need to create all three block objects at this time. First, if one wants to use one continuous buffer (rather than separate buffers for headers, data, and trailers and the use of scatter/gather IO [22]) the `RequestDispatcher` needs to write headers into the `OutputObject` before it is returned to the stub for marshaling. The `OutputObject` must agree with the `Connection` on the form of the internal buffer used between them. Secondly, we already mentioned the possibility, in RMI-IIOP, of having interceptors insert service contexts into headers that cause an overflow of the buffer when using GIOP fragmentation. In this case the `RequestDispatcher` would need to create a fragment message and give it to the `Connection` for sending even before marshaling begins.

At this point in our discussion we have seen how and when the main interfaces of the four blocks are created and how they are coordinated by the `RequestDispatcher` protocol block interface to marshal and send a request. We continue by examining how the reply is received and processed.

Wait for a Response. After the request is sent the client-side waits for a response from the server. The operation of waiting for a response is dependent on the protocol in use. `PEPt` gives the `RequestDispatcher` control over how to wait for a reply. An `HTTP RequestDispatcher` will simply block on a read of the `Connection` on which the request was sent. `RMI-IIOP` allows message multiplexing on a single `Connection`. Therefore it is necessary to demultiplex incoming replies. Since different reply messages (and possibly error and close connection messages) can arrive at any time, the `RMI-IIOP RequestDispatcher` would interact with a `ContactInfo` factory to create an appropriate protocol block `ProtocolHandler` object. The `ProtocolHandler` listens on the `Connection` for incoming messages (note, issues such as scalability using a "selector" for listening, or "non-listening" transports like Solaris doors are not discussed here). The `RMI-IIOP RequestDispatcher` would put itself to sleep waiting for the `ProtocolHandler` to signal that a matching reply has arrived. (Note: the `RequestDispatcher` and the `ProtocolHandler` taken together can be viewed as a form of "subcontract" [16].)

Get an Input Object for the Connection. When a reply arrives on the `Connection` we need to get an `InputObject` for the `Connection` so that we can read headers and the remote procedure's result.

When a reply arrives at the `Connection` it gives the raw bits of the reply to the `ProtocolHandler`. The `ProtocolHandler` examines the raw bits to determine the protocol in use (if the `Connection` is supporting multiple profiles). The `ProtocolHandler` then asks

ContactInfo to create an appropriate InputObject. (Note: a well-designed protocol will use the presentation block data types to read and write headers.)

In the RMI-IIOP case, after the InputObject has been created, the ProtocolHandler reads from it to determine the GIOP version, whether this is the first, continuing or last fragment of a reply or a complete (non-fragmented) reply, and to obtain the request ID. When the reply is non-fragmented or the first fragment of a reply the ProtocolHandler uses the request ID to find the matching request. It then gives the InputObject to the waiting RequestDispatcher and signals it to wake up to handle the reply. When the reply is a continuing or last fragment, the ProtocolHandler uses the request ID to find an existing InputObject (created during the first fragment). It gives the existing InputObject the raw bits of the reply. This forms a producer/consumer relationship between the ProtocolHandler and an existing InputObject.

Once the reply has been matched with a request, the RequestDispatcher will return the InputObject to the stub. The InputObject will be positioned to start reading the marshaled reply (the ProtocolHandler/RequestDispatcher having already read the header information). As noted above, if fragmentation is in effect there will be a private contract between the Connection, the ProtocolHandler and the InputObject such that as more fragments arrive for a particular reply those fragments can be passed to the internal buffers of the InputObject. The InputObject then serves the role of a shared buffer between the stub (consuming the InputObject) and the Connection/ProtocolHandler (filling the InputObject).

Unmarshal the Result and Cleanup Resources. The protocol block RequestDispatcher returns control and an encoding block InputObject to the stub when a reply has been received. The InputObject acts as a bridge between the encoding block and the presentation block.

After unmarshaling, before returning control to user code the stub signals the RPC infrastructure that it may clean up any resources used for this invocation. Example resources are fragment maps that map request IDs to InputObjects, the Input/OutputObjects and Connection used in the request, etc.

5.2 Server-Side Lifecycle

To save space, we will only mention key points regarding the server-side of remote requests. The steps to service a request are: 1: Accept a connection from the client. 2: Receive a request on the connection. 3: Get an input stream for the connection. 4: Find a tie and servant. 5: Use the input stream to unmarshal arguments. 6: Call the servant with the unmarshaled arguments. 7: Get an output stream for the connection. 8: Marshal the result or exception. 9: Send the reply. 10: Release any resources used in the remote call.

Accept a Connection, Receive a Request. The server Acceptor accepts a client's connection request and creates a PEPT transport Connection. When a request arrives the Connection gives the raw bits of the request to its associated Acceptor that acts as a factory for a ProtocolHandler. This gives Connections the ability to handle multiple protocols by delegating the creation of the ProtocolHandler to the Acceptor, which may decode some portion of the initial raw bits to determine the protocol in use and create the appropriate handler.

Get a Request Input Object and Umarshal Header. Once the Acceptor has determined the protocol in use it gives control to the ProtocolHandler that then asks the Acceptor to act as a factory for an InputObject for the Connection.

The ProtocolHandler reads message headers from the InputObject to determine the intent (i.e., type) of the message. The ProtocolHandler may use header information to determine which RequestDispatcher to use to handle the request or it may delegate this determination to the Acceptor. The ProtocolHandler is logically separate from the RequestDispatcher so that if any errors occur during header processing (e.g., header unmarshaling errors) it can form an error reply appropriate for the protocol.

Note that the Acceptor is the server side factory for Connections, ProtocolHandlers, RequestDispatchers, InputObject and OutputObjects. Thus, Acceptor is the primary server-side pluggability point in the PEPT architecture (similar to ContactInfo on the client-side).

Find a Tie, Unmarshal Arguments, Call Servant, Marshal Result. The ProtocolHandler gives control to the RequestDispatcher that finds the appropriate type-specific tie and servant. A tie unmarshals the arguments, calls the servant, then marshals the results.

The presentation block tie gets an OutputObject by interacting with the protocol block RequestDispatcher which, in turn, will interact with the transport block Acceptor and Connection to obtain the correct type of OutputObject. The protocol block may write reply headers into the OutputObject (which may result in fragments of the reply being sent on the Connection). Note that RMI-IIOP's ResponseHandler can be viewed as a standard interface to RequestDispatcher.

Send Reply and Cleanup. When marshaling is complete, the presentation block tie signals the protocol block's RequestDispatcher and ProtocolHandler to resume control. The protocol block takes the encoded framed data and sends it to the client on the transport block Connection: Any resources used while processing the request may then be cleaned up under control of the RequestDispatcher and/or ProtocolHandler.

5.3 Lifecycle Summary

We have shown that the steps taken to invoke and service a remote procedure are essentially the same regardless of the specific presentation block types and APIs, en-

codings, protocols and transports used. The following tables summarize the blocks and interfaces used at each step. Note that fragmentation may happen any time an output or input object is written or read. We indicate what blocks are involved in fragmentation in rows labeled "(fragmentation)".

Table 1. Client-side steps

Operation	Presentation	Encoding	Protocol	Transport
<i>getConnection</i>			ReqDispatcher	ContactInfo
<i>getOutputObj</i>	stub		ReqDispatcher	ContactInfo
<i>marshalHeader</i>		OutputObject	ReqDispatcher	
<i>(fragmentation)</i>		OutputObject	ReqDispatcher	Connection
<i>marshalArgs</i>	stub	OutputObject		
<i>marshalDone</i>	stub		ReqDispatcher	
<i>sendRequest</i>		OutputObject	ReqDispatcher	Connection
<i>waitForReply</i>			ProtocolHandler	Connection
<i>getInputObject</i>			ProtocolHandler	ContactInfo
<i>unmarshalHDR</i>		InputObject	ReqDispatcher	
<i>(fragmentation)</i>		InputObject	ProtocolHandler	Connection
<i>unmarshalReply</i>	stub	InputObject		
<i>returnResult</i>	stub			
<i>cleanup</i>	stub		ReqDispatcher	

Table 2. Server-side steps

Operation	Presentation	Encoding	Protocol	Transport
<i>getConnection</i>				Acceptor
<i>receiveReqBits</i>			ProtocolHandler	Connection
<i>getInputObject</i>			ProtocolHandler	Acceptor
<i>getReqDispatch</i>			ProtocolHandler	
<i>unmarshalHDR</i>		InputObject	ReqDispatcher	
<i>(fragmentation)</i>		InputObject	ProtocolHandler	Connection
<i>getTie/Servant</i>			ReqDispatcher	
<i>unmarshalArgs</i>	tie	InputObject		
<i>callServant</i>	tie			
<i>getOutputObj</i>	tie		ReqDispatcher	Acceptor
<i>marshalHeader</i>		OutputObject	ReqDispatcher	
<i>(fragmentation)</i>		OutputObject	ReqDispatcher	
<i>marshalResult</i>	tie	OutputObject		
<i>marshalDone</i>	tie		ReqDispatcher	
<i>sendReply</i>		OutputObject	ReqDispatcher	Connection
<i>cleanup</i>			ReqDispatcher	

6 Conclusions and Future Work

RPC specification and implementation could benefit from isolating the key concepts into a core architecture. To this end, we propose PEPT, a four-block design that decomposes RPC systems into presentation, encoding, protocol, and transport blocks. With such an approach, one aspect of the RPC may evolve without disturbing the others. In other words, when an alternate encoding, protocol or transport is desired there is no need to create another presentation block. Or, alternatively, a new presentation block can reuse existing protocols, encoding and transports.

The PEPT architecture has been used in a commercial CORBA product [11]. That same implementation has been used to prototype a system that supports RMI-IIOP stubs and ties dynamically switching between IIOP and SOAP/HTTP. The core RPC architecture can serve as the basis for understanding, designing, implementing, maintaining and reusing RPC systems.

Although this work makes a fair case for PEPT, space limitations have precluded treatment of important issues such as transactions, security, threads and thread pools, and connection caches. We need to show more detail of each block and other supporting interfaces not shown in this paper, as well as showing finer-grained detail how specific systems are implemented using the PEPT architecture. In particular we need to show how InputObject and OutputObject properly isolate encoding changes from the rest of the system. This is especially important when switching from a self-describing encoding to a binary encoding. The self-describing encoding more gracefully handles extra or missing data. The binary encoding is more compact but may require more agreements between parties. It will be illuminating to measure the throughput and latency for different encodings to answer questions such as: what is the cost of redundancy? What is the cost of self-describing data? We would like to show that invoking SOAP-based web services from a general RPC system need not be difficult. The only real complexity is isolated to where it should be - in the data that is sent back and forth.

We need to show how the protocol block and encoding block deal with the situation where fragments of a request are sent, causing a reply (complete or fragment) to be received even before marshaling is complete. This can happen in RMI-IIOP when the server-side detects an error early in processing or forwards the request to a different object.

Since most underlying protocols are asynchronous we think that PEPT can also serve as the basis for messaging systems, thus unifying our thinking and implementation of those systems. The primary work that needs to be done here is to partition the presentation layer into two dimensions: synchronous versus asynchronous, and message API versus method call. We also need to investigate how the blocks support issues such as messaging store and forward.

Acknowledgments. I would like to thank my friend and colleague Ken Cavanaugh at Sun Microsystems for reviewing this paper and, more importantly, being the first person to understand and use the PEPt architecture. Much of the experience and wisdom embodied by PEPt comes from many years of working with Ken. David Ungar, also at Sun Microsystems, asked pointed questions which helped focus the presentation. I had useful interactions with Peter Walker, Hemanth Puttaswamy, Masood Mortazavi, Nick Kassem, Mark Hapner, Roberto Chinnici, Phil Goodwin, Anita Jindal, Arun Gupta and David Heisser (all at Sun) during the preparation of this paper. Dan Miranker at UT Austin encouraged me to write.

References

1. Nelson, B.J.: Remote procedure call. Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa. (1981)
2. The Open Group: DCE. <http://www.opengroup.org/pubs/catalog/dz.htm>.
3. Carr, H.: Distributed C++. Ph.D. thesis, University of Utah (1994)
4. Kesselman, C., Mani, C.K.: Compositional C++: Compositional Parallel Programming. Caltach (1992) <http://caltechcstr.library.caltech.edu/documents/disk0/00/00/01/05/>
5. Microsoft: DCOM. <http://www.microsoft.com/com/tech/DCOM.asp>
6. Object Management Group (OMG): Common Object Request Broker Architecture (CORBA). http://www.omg.org/technology/documents/formal/corba_iiop.htm
7. Sun Microsystems: Remote Method Invocation. <http://java.sun.com/products/jdk/rmi/>
8. OMG: Java to IDL Mapping. <http://www.omg.org/cgi-bin/doc?ptc/02-01-12>
9. Winer, D.: XML-RPC. Userland (1999) <http://www.xmlrpc.com/>
10. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. World Wide Web Consortium (2000) <http://www.w3.org/TR/SOAP/>
11. Sun Microsystems: Corba Technology and the Java 2 Platform, Standard Edition. <http://java.sun.com/j2se/1.4.1/docs/guide/corba/index.html>
12. Schmidt, D.C.: The ADAPTIVE Communication Environment (ACE). <http://www.cs.wustl.edu/~schmidt/ACE.html>
13. International Organization for Standardization: Reference Model of Open Distributed Processing. <http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20696>
14. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. World Wide Web Consortium (2001) <http://www.w3.org/TR/wsdl>
15. jini.org: Jini Davis Project. <http://davis.jini.org/index.html>
16. Hamilton, G., Mitchell, J.G., Powell, M.L.: Subcontract: A Flexible Base for Distributed Programming. Sun Microsystems (1993) <http://research.sun.com/research/techrep/1993/abstract-13.html>
17. Myers, J.: Simple Authentication and Security Layer (SASL). Internet Engineering Task Force (1997) <http://ietf.org/rfc/rfc2222.txt?number=2222>
18. Rose, M.T.: BEEP. <http://beepcore.org/>
19. Boecking, S.: Object-Oriented Network Protocols. Addison Wesley (2000) <http://www.aw.com/catalog/academic/product/1,4096,0201177897,00.html>

20. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Volume 2. Patterns for Concurrent and Networked Objects. John Wiley and Sons, Ltd (2000) <http://siesta.cs.wustl.edu/~schmidt/POSA/>
21. Sun Microsystems: Solaris Doors.
<http://docs.sun.com/db/doc/806-0630/6j9vkb8d1?a=view>
22. McKusick, M.K., Bostic, K., Karels, M.J., Quarterman, J.S.: The Design and Implementation of the 4.4BSD Operating System. Addison Wesley (1996) 2.6.5 Scatter/Gather I/O.
http://www.freebsd.org/doc/en_US.ISO8859-1/books/design-44bsd/x355.html#AEN458

Modelling Organic Adaptable Service-Oriented Enterprise Architectures

George R. Ribeiro-Justo^{1*} and Tereska Karran²

¹Cap Gemini Ernst Young, UK
george.justo@cgey.com

²Cavendish School of Computer Science, University of Westminster, London, UK
karrant@wmin.ac.uk

Abstract. Developing systems that are capable of monitoring and learning about themselves, and thereby rapidly react to changes in their environment, has become essential to any enterprise. The search for solutions to this complex and challenging problem has stimulated both researchers and practitioners. This paper attempts to connect the views of the researchers and practitioners by relating the state-of-the-art approach proposed by the Complex Organic Distributed Architecture (CODA), with Cap Gemini Ernst & Young's vision of adaptive architecture. CODA applies cybernetic concepts such as self-organisation, self-regulation and viability to derive an intelligent architecture, which can react to failures in achieving its objectives and proactively search for successful patterns of behaviour. The paper presents a model for developing adaptive enterprise systems using CODA, which at the same time satisfies Cap Gemini Ernst & Young's characteristics for adaptability. The paper also briefly describes how the model can be implemented using current enterprise infrastructures.

1 Introduction

The recent economic crisis, that resulted mainly from the dotcom bubble burst, has forced enterprises not only to reduce their IT expenditures but more importantly to emphasise the need for them to be more agile and adaptable to changes in their ecosystems (network of value relationships partners, including customers, suppliers, dealers, competitors). Adaptability must therefore be seen from a holistic perspective, encompassing all levels of the enterprise architecture from the infrastructure to the business models.

Very few proposals follow this holistic approach. One exception is the Cap Gemini Ernst & Young (CGE&Y) concept of *Adaptable Architecture* [Schekkerman 2002], which is based on a long tradition of architecture development of enterprise architectures using IAF (*Integrated Architecture Framework*) [Kornak 2002].

* The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied of Cap Gemini Ernst & Young.

The essence of the IAF approach to architecture, which has been adopted by industry-standard groups such as the Institute of Electrical and Electronics Engineers (IEEE), is its business-driven model. “Business that fails to adequately utilize technology will not achieve its potential. Conversely, technology that is not aligned with governing business objectives will fail to adequately support the business” [Kornak 2002]. The architecture must translate business strategies and principles into IT strategies and principles, which in turn define how the business concepts will be implemented and all services and products that are used.

The adaptive architecture is also derived from the IT strategy and principles but it focuses on how a business should evolve its IT business systems in order to successfully thrive in the future business environment [Kornak 2002]. Adaptive architecture is about making a business more agile, making it easier to react and adapt to changes in its environment, from business opportunities or threats to the emergence of new technologies.

The problem of adaptability has been studied for some time and we have previously investigated several aspects of reconfigurable distributed systems that can support adaptability [Ribeiro-Justo 1999]. The premise is that dynamic changes in the system software architecture at runtime can increase adaptability. Further study has also shown that those changes can provide increased adaptability when they support monitoring and tuning of the system’s non-functional requirements [Ribeiro-Justo 2002]. After all, the architecture incorporates and is home to all non-functional requirements. Satisfying its non-functional requirements is not only essential for a system’s success but also for its degree of adaptability. For instance, a system that is neither scalable nor robust can hardly be seen as adaptable.

Other lines of investigation have focussed on other aspects of the adaptability. This reinforces our conviction that a holistic approach is needed to support adaptability. Consequently, a plethora of research is needed. Adaptable middleware research aims principally at supporting dynamic customisation in order to provide the flexibility required for adapting the middleware to cope with changing quality of service or resource requirements [Agha 2002].

Active entropy is an architectural paradigm that mimics the continuous, online feedback that is the hallmark of living organisms. Introspection devotes computational resources to observing system behaviour, then adapts the system accordingly [Kubiatowicz 2003]. The principle is that introspection adds information to the system in order to improve future behaviour. The IBM autonomic computing follows this principle. Autonomic computing focuses on the ways computers can carry out necessary tasks automatically, similarly to the way the human autonomic nervous system regulates breathing and other functions [Paulson 2002]. The system applies AI techniques to learn about the host computing environment and how it changes. If the system is having problems, the self-healing technology can send instructions as to how to implement a solution.

The autonomic approach is similar to the way CODA (The Complex Organic Distributed Architecture), presented in this paper, works [Ribeiro-Justo 2001, Karran 2003]. CODA applies the cybernetic concepts proposed by the Viable System Model (VSM) [Beer 1985]. The foundations underpinning the VSM are that all living systems are composed of a series of autonomous sub-systems, each having self-organizing and self-regulatory characteristics. They have within them the capacity to

adapt to changes in their environment and to deal with the complexity that is relevant to them [Espejo 1997].

The CGE&Y vision of adaptive architecture relates to all those concepts of adaptability but in essence it defines three key characteristics that an adaptive architecture should present:

- **Autonomy:** the capability to make decisions about actions based on circumstances without the need for human intervention;
- **Reactivity:** the ability to make dynamic and appropriate responses to unpredicted circumstances or environments;
- **Reactivity:** the anticipation of future requirements to order the process in the correct or optimum manner;

CODA [Karran 2003] satisfies the above criteria, and one of the main objectives of this paper is to demonstrate how to model an adaptive enterprise architecture applying the CODA principles.

In the next section, we summarise the cybernetics concepts behind CODA adaptability and Section 3 introduces the model for adaptive enterprise architecture. An application of the model is presented in Section 4. Section 5 briefly describes how the model can be implemented using current enterprise infrastructure technology. Finally, Section 6 presents the conclusions of this paper and directions for future work.

2 Overview of CODA Foundations

A system is a bounded collection of three types of entities: elements, attributes of elements, and relationships amongst elements and attributes. Both attributes and relationships are characterised by functions called *variables* [Waelchi 1996].

A systemic view of the achievement of organisational goals is accomplished in a three-step program: selection of formal variables inside and outside the organisation, prescription of acceptable values for each variable, and expenditure of managerial effort to generate and transmit information required to fit the key values into the desired ranges and keep them there [Waelchi 1996].

A *viable system* is the one capable of independent existence [Beer 1985]. To survive, a viable system needs not only a capacity to respond to familiar disturbances, but potentially to respond to unexpected, previously unknown disturbances. The latter is the hallmark of viable systems. It gives them the ability to adapt to changing environments [Espejo 1997]. Beer shows us how organisational structure – the necessary requirement for achieving purposes – can be given its essential property: viability.

A precise measure of (systemic) complexity has been defined as *variety*, meaning the number of distinguishable elements in a system, or by extension, the number of distinguishable *systemic states*. The state of a system at any time is the set of values held by its variables at that time. The notion of a *coenetic* variable explains the range of the variety of environmental circumstances and type of regulatory responses [Beer 1985].

The *Law of Requisite Variety* established by Ashby [Waelchi 1996] is used as a heuristic to develop criteria of effectiveness. Ashby's Law describes the conditions under which a complex system can be externally controlled. The values of certain variables (essential variables) must remain within limits for the system to continue in existence. A controller has requisite variety i.e. has capacity to maintain the outcomes of a situation within desirable states (the target set) only if it has the capacity to produce responses to all those disturbances that are likely to take the outcomes of the situation out of its target set [Espejo 1997].

The good news is that many variables display equilibrium—that is; the tendency towards a small range of values. This quality is known as goal-seeking behaviour. The control operates at two levels: the first one is required to control the value of the essential variable and the second one (operational) is used to control the system according to its purpose. The actual mechanism of control, of creating the desired changes in the values of the variables in the controlled system, is the flow of information from the controlling system to the controlled system [Waelchi 1996]

Based on Ashby's law, Beer [Beer 1985] defines the concept of the Viable System Model (VSM). The VSM is devised in terms of sets of interlocking Ashbean homeostats. An individual operation would depict as homeostatically balanced with its own management on one side and its market on the other. In the VSM, the homeostats requisite variety is applied to the block of variety homeostatically related to the channels carrying information between them and to the transducers relaying information across boundaries.

Every viable business can be seen as a collection of viable businesses together with a system that manages the relations between these viable lower-level activities so it is viable as a whole. Part of the variety is then pushed down to the next level and the remaining task is manageable. The proper tasks at each level are to formulate adaptive strategies and identify synergistic opportunities of the level immediately below. Key to the VSM theory is establishing that in any viable systems, there are five necessary and sufficient sub-systems involved in any organism or organisation [Espejo 1997], as illustrated in Fig. 1:

- **Implementation:** This subsystem encompasses the primary activities performed by an organisation, such as the production of products and services.
- **Co-ordination:** A viable system has a sub-system that co-ordinates the operations of its primary activities.
- **Control:** a viable system requires supervisory control supported mainly by a monitoring channel and the provision of an exception reporting system.
- **Intelligence:** This function focuses on the future, concerned with planning ahead.
- **Policy:** This system defines the direction, values and *raison-d'être* of the organisation. The policies are based on selective information provided by the Control and Intelligence systems.

These sub-systems are organised as layers in CODA taking into account architectural concept of enterprise systems. Details of the layers will be presented later, when we show how the layers can be modelled. The theory behind CODA has already been demonstrated elsewhere [Ribeiro-Justo 2001, Karran 2003], therefore the focus of this paper is to present a model for developing adaptive enterprise architectures based on CODA.

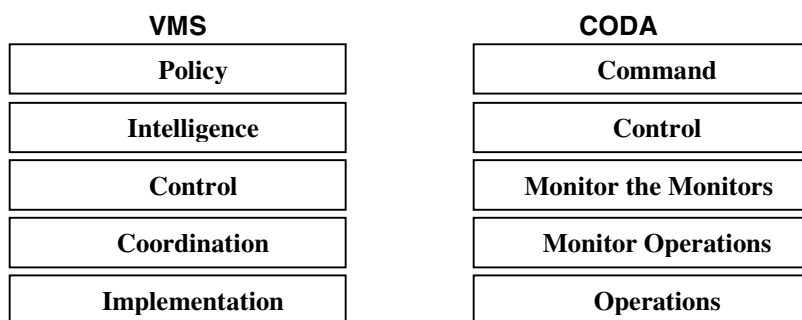


Fig. 1. Relationship between the VSM subsystems and the CODA layers.

3 Modelling Adaptable Enterprise Architectures with CODA

At the heart of the CODA model is the concept of a *Role*, as shown in Fig. 2. A *Role* denotes an *agent* playing a specific role in the system. The reason for using the concept of a role rather than an agent is to emphasize the idea that a role has access to certain tasks. This is important to enable the system to control what can be done at critical times, for instance when resources are scarce, and consequently certain tasks should not be executed. This is key to the concept of adaptability.

A role can be seen as an active service, which performs tasks and operates on the goal of reaching critical successor factors (CSFs). A CSF is modelled as a measured objective. The CSFs define the SLAs (Service Level Agreements) of the service. For example, 80% of gold users should be served and similarly 50% of silver users, as shown in the example in Section 4.

The role's tasks are constrained by its filters, which determine the information that can be passed between layers (this concept will be discussed in more detail later), but more importantly establish the current CSFs the role should achieve. The filters are another mechanism used to support adaptability. For example, if the system is failing to provide the service to the gold users, it may block access to other types of users. To avoid unnecessary failures of CSFs, the filters of a role can be adjusted to block that service. Similarly, when the system is underused, extra services can be added. In summary, the filters define the operating parameters of the role based on its CSFs.

Short and long time memory are denoted by a *Wrapper*, which stores both current and historical events of the role. These events indicate the successes and failures of the role and are crucial to enable the system to adapt. This section has briefly introduced the key concepts of the model. The next sections will discuss how these concepts are used to support the characteristics required for adaptability.

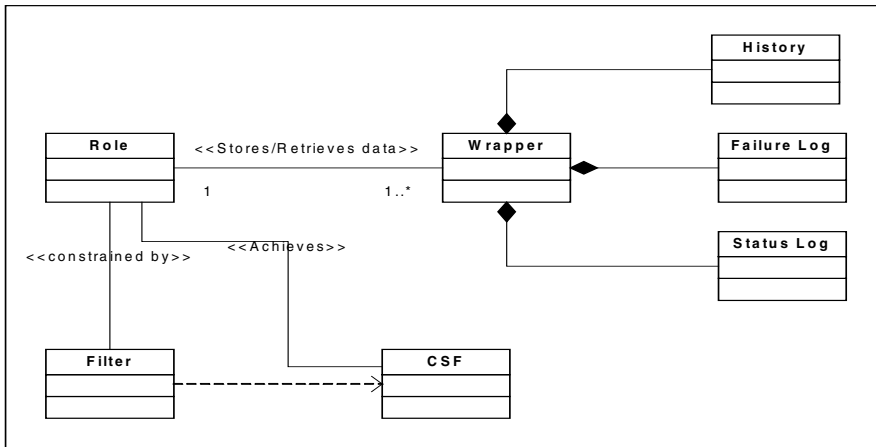


Fig. 2. Coda component model

3.1 Modelling Autonomy

The key principle behind the concept of role is autonomy. A role should perform its tasks autonomously, provided its CSFs are satisfied. If a CSF fails, the role will need assistance from other roles, possibly a higher-level role, which should have access to information and tasks not normally available to it. The higher-level role should be able to adjust the operational parameters (specified as CSFs and filters) of the failed role, to enable it to adapt to the conditions that are causing the failure. If the higher-level role also fails to achieve its CSFs, it similarly alerts another higher layer role. As will be shown later, the chain of roles can extend to as many layers as there are in the system and only if it is not possible for the system to decide what to do, will human intervention be needed.

3.2 Modelling Reactive Behaviour

A role can be seen as a control component, which tries to achieve its objectives following the current operational parameters. If the role is unable to respond using those parameters, an *alert* is generated. The alert basically corresponds to a ‘help call’ to a ‘more intelligent’ component (higher layer role), which may assist the failed role in reacting to the event that has caused a CSF failure. CODA *feedback loop* mechanism allows the higher layer role to respond by resetting the failed role’s operational parameters by adding or removing filters, as illustrated in Fig. 3.

In extreme cases, CODA may even shutdown the system or disable access from groups of users, if this proves necessary for the system survival when it is under threat. For instance, when the system runs out of resources or it recognizes a hacker.

The concept of feedback loop in control systems is well known [Beer 1985]. The CODA concept of feedback loop is more sophisticated because of its notion of systemic learning. Rather than just adjusting the operational parameters to cope with environment changes like traditional control systems, CODA uses its memory, the

information provided by the wrappers, to identify the successful operational parameters. In this sense, the model presents a level of intelligence that is not encountered in typical control systems. In addition, because of the amount of information collated by enterprise information systems, the area where CODA is most successfully applied, the capability for learning is further enhanced even in comparison with other kinds of intelligent systems [Kayam 2003]. This issue will be discussed in more detail when modelling CODA layers.

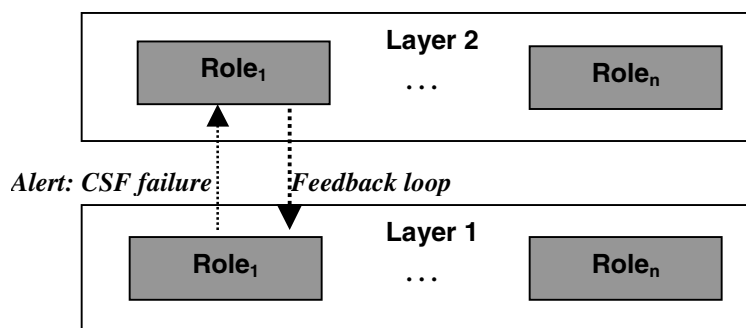


Fig. 3. Coda reactive model.

3.3 Modelling Proactive Behaviour

In the previous section, the use of CODA concept of memory (current and historical data) was important to help the system to react, via the feedback loop, to unpredicted behaviour caused mainly by failure in CSFs. Historical data has another key function in the model to support proactive behaviour. For instance, if the network has spare bandwidth a silver user may be offered broadband access, where in general that type of service is only provided for gold users.

An enterprise architecture modelled using CODA needs to make use of predictive techniques usually available in business intelligence [SAS 2001]. This is essential for the system to predict its resource usage and successful services and thereby be proactive in adjusting CSFs and operational parameters. In addition, it is even possible to model emergent behaviour where the system can identify behaviour not previously known. For instance, the system may identify new user patterns or service usage.

3.4 Modelling Layers

The concept of layers is fundamental in CODA. The principle is to separate the levels of intelligence of the system and also to categorise the data used by the system. In general, as illustrated in Fig. 2, the structure of each layer is basically the same, which means that a system with an infinite number of layers would be possible. However, CODA theory recommends a maximum five layer model, following the VSM theory, as described in Section 2. CODA layers, however, focus not only on functions

enterprise systems must perform, following the VSM principles, but also on the data structures, as described below (refer to Fig.1 to relate CODA layers to VSM levels):

- **Operations:** This layer deals with simple linear data, which usually corresponds to transaction processing and business operations of a typical enterprise architecture. The operational data warehouse usually links together data from databases in several locations. This layer can be modelled as an existing system, which will provide the core data of the system.
- **Monitor Operations:** In this layer, the data is often dimensional and aggregated. For instance, data is organised by time or group. This layer is responsible for monitoring business operations. Roles in this layer can react quickly to operational failures by filtering requests, which may affect CSFs.
- **Monitor (the) Monitors:** This layer deals with multidimensional data and provides capability for analysing trend behaviour. At this level, business operations are monitored in terms of external trends. It is possible to react more effectively by analysing the historical data from various levels of aggregations such as type, time and locations. We can identify the types of services requested at specific locations at certain times. The system will therefore be able to provide more successful filters. For instance, when a silver user logs on to the system during peak time in the middle of London, broadband service may not be available, if CODA goal is to serve gold users. This shows that this layer can also provide a degree of proactive behaviour, by predicting trends.
- **Control:** This layer should be able to “learn” about simple emergent behaviour, trends and forecasts and be able to run predictions and simulations automatically. Although the Monitor the Monitors layer can predict certain trends, this is actually modelled at this layer.
- **Command:** This is the highest layer, which should be able to deal with any variety not treated by the lower layers [Waelchi 1996]. This means being able to recognise new threats and opportunities. Here we deal with strategic and long-term trends. This layer should make use not only of its historical data but also of outside data sources in the architecture ecosystem. In addition, the lower layers make their decisions on data derived from operational data and this layer will need access to data sources outside the operational system.

Although the above five layers are recommended for a complete adaptable enterprise architecture, it is possible to develop successful adaptable systems with only three of these layers, provided the Monitor the Monitors layer has predictive capabilities. It is important to observe that in fact most so-called ‘adaptive’ or reconfigurable systems only present two layers, which can be related to the two lowest layers of CODA [Narayanan 2000]. However, these systems do not apply the concept of CSFs and filters, which are necessary for adaptability.

4 Modelling an Adaptive Mobile Enterprise Architecture

Serving a customer is the core tenet of any organisation that provides a product, a service, or information. Whether or not it is a business for profit, the most central business function of any organisation is to transfer its offerings to those that desire

them. All these organisations, commercial, government or educational institutions, must interact with their clients one way or another, which requires the allocation of scarce resources [Kornak 2002].

This section describes an adaptive service-oriented enterprise system modelled using CODA. The system is a 3G mobile network. The system is similar to that presented in [Karran 2003] but this paper concentrates on service-oriented adaptability rather than on the network reconfiguration. The overview of the adaptive mobile network is shown in Fig. 4.

A variety of services are offered to the customer, ranging from voice and text to video-on-demand. Each of these services requires a distinct amount of resources, including access to lines and bandwidth. In addition, the quality of service, which includes error-rates and failures, can also vary. Key to every service provider is the concept of service-level agreement (SLA), which establishes the contractual obligation in terms of the quality of service expected by the customer. In CODA, we model SLAs as CSFs and the system using introspection will monitor them together with the available resources in order to achieve its goals. The SLAs are associated with roles, which denote service providers, similar to that when using IAF. Customers are categorised into types, such as gold, silver and bronze, each with different SLAs. The following section describes the details of the model.

4.1 Modelling Layers and Role

In this case study, we assume that three layers will be sufficient to provide the adaptability required by the application, as illustrated in Fig. 5. Although CODA recommends a complete adaptable system to contain five layers, simpler systems do not require such a complete architecture. It is essential to ensure that the architecture is sufficiently independent from human intervention, reactive and proactive, as previously stated. Provided the Monitor the Monitors layer is capable of predicting future behaviour and can analyse complex aggregated data, the architecture will satisfy the essential requirements for adaptability.

The Operations layer is modelled using three roles, representing the mobile station (MS), the base station (BS) and the mobile station controller (MSC). For simplicity, as shown in Fig. 5, the BS is modelled as both the base station transceiver and the base station controller. The naming convention we follow is to prefix the role name with the layer name. For instance the MS role at the operations layers is named *OpMS*. We do not model the concept of customer type. Instead we apply the concept of Role sub-typing to the MS object. This results in the gold, silver and bronze MS roles respectively as shown in Fig. 5.

The roles perform tasks and in doing so may request services from other roles. The first service the *OpMS* provides to the user is authentication, when the user turns on his/her mobile device. The *OpMS* requests the service of the *OpMSC* to carry out the authentication. If successful, the customer will be able to access the mobile network and request services. Observe that in a real mobile network, the request should go through the BS. The *OpMSC* provides filters for the *OpMS*, depending on the customer type and the mobile device used, the customer location (the BS the customer is connected to), and time of the day. More details of the filters will be presented in Section 4.4.

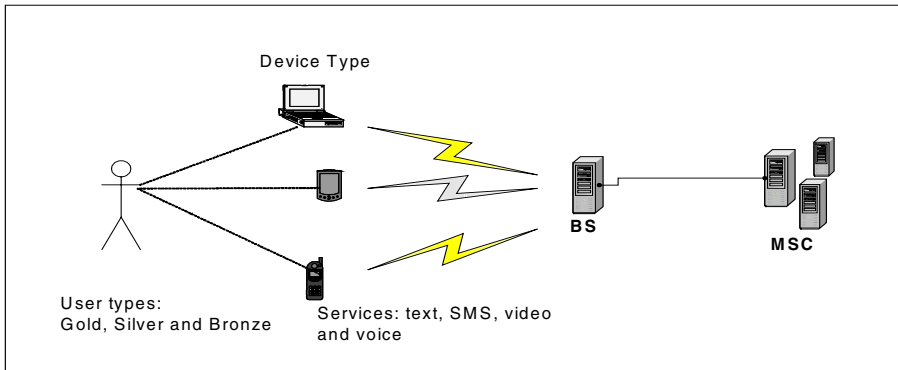


Fig. 4. Adaptable mobile network

After being successfully authenticated, the user can make calls (request services). These requests will be sent to the OpBS, which checks if it has sufficient resources to support that service, based not only on the actual availability of resources but also on its CSFs, as we describe in the next section. The OpBS uses the OpMSC to check whether the called device is connected to the network and is able to receive the call. Successes and failures are stored in the OpBS wrapper.

Observe that, as the only critical resources required for the successful operation of the system are provided by the BS, only the MBS role is modelled at Monitoring layer. The MBS aggregates the data about the BS operation time and user type. The objective is to analyse resource requirements at different times of the day for different types of user. This will help to reduce offers of certain services for certain user types at certain times of the day. It may also identify threat situations when the OpBS is in danger of running out of resources. In this case, it may proactively react by sending a new filter to the OpBS to block any new calls. Conversely, it may identify opportunities when there is spare bandwidth and special services may be offered to customers in addition to those defined in their current contract. For instance, a silver customer with a 3G device may be offered the opportunity to send videos. The MBS wrapper is an aggregation of all OpBS roles' wrappers.

The main goal of the MMMSC role at the Monitor the Monitors layer is to manage the BS but, since there are no MS and MSC at the Monitoring layer, it also monitors these roles in order to control the number of users authenticated and active (making calls) and their success rates. However, the MMMSC's most important task is to analyse the operations and monitoring data in order to recognise success and failure patterns. For example, failure by service, device and user type over time. If a certain device failure rate at a certain time is too high it may not filter that service.

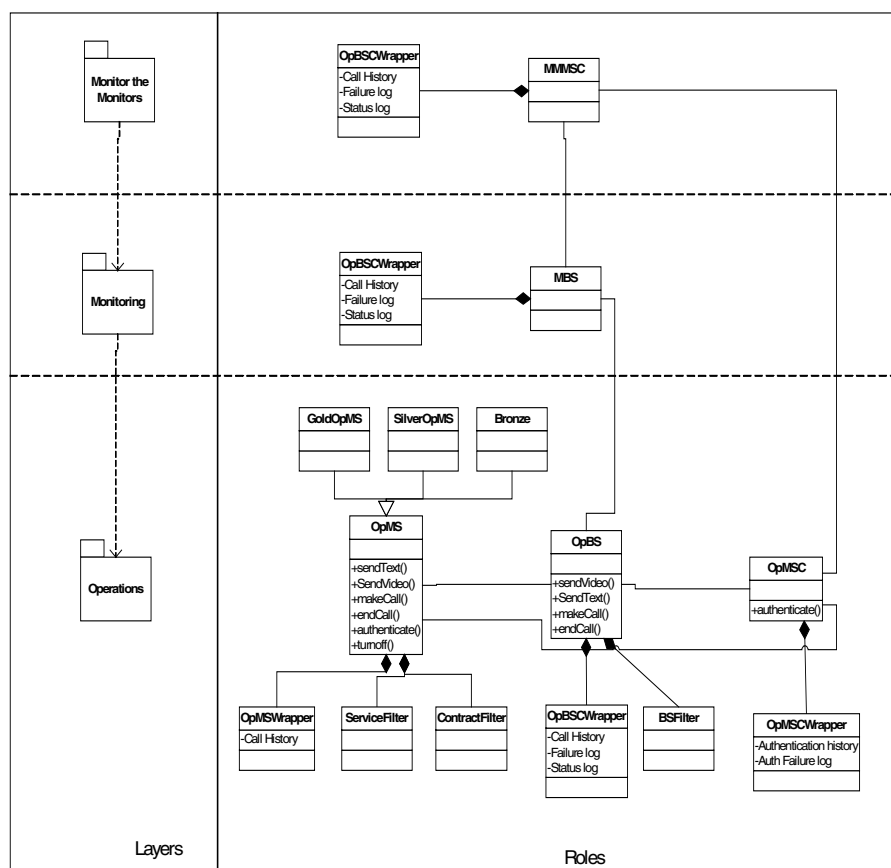


Fig. 5. UML model of the adaptable mobile architecture

4.2 Modelling Critical Success Factors

CSFs are measurable goals. They are the parameters that enable the system to analyse itself and are key for reactive behaviour and autonomy. As long as the CSFs are satisfied the role will not depend on the intervention of others.

Simple roles may not require explicit CSFs, as their filter may constrain their behaviour in such a way that the CSFs will be satisfied by default or are carried out indirectly by other roles. In our example, we do not assign any CSF to the OpMS or to the OpMSC (since we assume that there is no limit on the number of users that can be authenticated). This does not mean, however, that all users will be able to make calls. The BS, however, has a limited number of resources to attend the customers.

In a service-oriented architecture, the CSFs are principally associated with the quality of service offered to the customer. The specific CSFs will depend on the system requirements and are derived from the IT principles. The BS will try to service calls according to the user type with certain quality. We specify, for instance, that it

should achieve success in serving 80% of gold customers, 70% of silver customers and 60% of bronze customers calls. We can further specify the error rate (quality of service) of 5%.

To ensure the viability of the system, survival CSFs are also specified. These indicate the danger level, when the number of lines and bandwidth available are below a certain threshold, for instance, 10%.

CSFs are also categorised according to the dimensions in which the system will be self-analysed. For example, CSFs are specified for different time bands, when time is an important issue for the service provision.

4.3 Modelling the Feedback Loop

What happens when a CSF fails? As previously explained, the associated higher layer role needs to reset the failed role by virtue of its filters.

When an OpBS fails to offer call services, it sends an alert to the MBS, which sends a new filter to the OpBS to reduce the service offered to bronze customers. If that does not solve the problem it reduces the services for silver users and finally to gold users. If the number of failures at the MBS passes its CSFs, then it must alert the MMMSC.

The MMMSC first tries to define new filters to newly authenticated customers and if this does not solve the problem it may block all services for newly authenticated customers until the system stops failing.

4.4 Modelling the Filters

The MS service requests are initially filtered during authentication. The OpMSC specifies which services are available for that customer type during that time. Two types of filters are actually associated with the MS, as shown in Fig. 5. The contract filter specifies the services that may be available for the customers according to their contract whilst the service filter defines the services that are actually available for the customer at a certain time band

The BS filter specifies the services that the BS can offer at that time. These filters may change for the next time band or if the system cannot cope with the service requests. To demonstrate proactive behaviour another filter allows the MSC to offer special services during authentication. As previously explained, these services can be offered if there is spare capacity in the system.

5 Implementing CODA Models

The CODA run-time environment leverages adaptable infrastructure technologies available to enterprises such as application server, message bus and enterprise system in order to simplify deployment and management. The J2EE (Java 2 Enterprise Environment) [Kassem 2000] CODA environment is shown in Fig. 6.

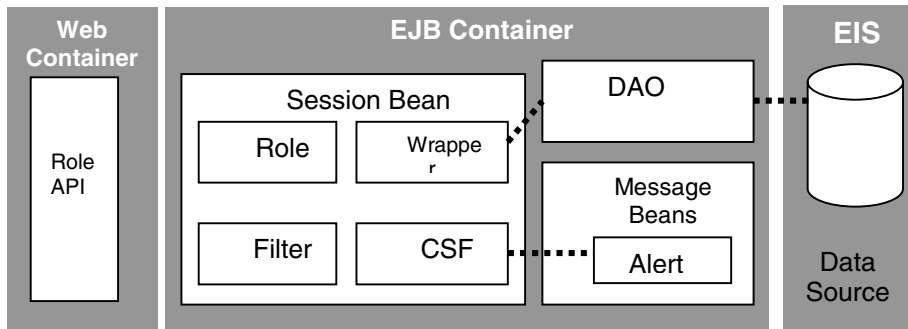


Fig. 6. J2EE Coda environment

The principal CODA component, which combines the Role and its elements (Filters, Wrappers and CSFs), is implemented as a session bean. This enables it to take full advantage of the main characteristics of the application server, such as scalability, availability, fail-over and security. Communication between Roles, in particular CSF alerts, is carried out using message queues. These are implemented using JMS (Java Message Service) [Kassem 2000] and a message bean is associated with every Role. Data access objects (DAOs) manage the access to operational data and most importantly the access and generation of historical data. These objects usually access data sources using JDBC (Java DataBase Connectivity) but Java connectors are also used [Kassem 2000].

The Role's services are exposed to the clients via its API as a delegate object. These delegate objects are used by portal components. We are also investigating modelling these objects as web services.

6 Conclusions

Developing systems that are capable of monitoring and learning about themselves and rapidly react to changes in their environment have become essential to any enterprise. The search for solutions to this complex and challenging problem has stimulated both researchers and practitioners. This paper has attempted to connect these two fields by relating state-of-the-art approach, CODA, with CGE&Y's vision of adaptive architecture.

CODA applies cybernetic concepts such as self-organisation, self-regulation and viability concepts to derive an intelligent architecture, which can react to failures in achieving its objectives and proactively search for successful patterns of behaviour.

In this paper we attempted to demonstrate how these concepts could be modelled into concrete adaptive enterprise systems.

Further empirical investigations are being carried out to evaluate CODA systemic learning capability. We are also developing a new version of the CODA model using BEA WebLogic's latest enterprise platform to explore the use of business process modelling tools and web services.

References

1. S. Beer, *Diagnosing The System for Organisations*, Wiley, 1985.
2. G. A. Agha, Special Issue on Adaptive Middleware, *Communications of the ACM*, Volume 45, Number 6, June 2002.
3. R. Espejo and A. Gill, *The Viable System Model as a Framework for Understanding Organisations*, Phrontis Limited & SYNCHO Limited, 1997.
4. T. Karran, K. Madani and G. R. Ribeiro-Justo, Self-Learning and Adaptive Systems: The CODA Approach, in *Software Defined Radio: Architectures, Systems and Functions*, edited by M. Dillinger, K. Madani and N. Alonistioni, John Wiley & Sons Ltd., 2003.
5. N. Kassem, *Designing Enterprise Applications with the Java(TM) 2 Platform (Enterprise Edition)*, Addison-Wesley, 2000.
6. Kayam and S. Bailey, *Intelligent Architectures for Service-Oriented Solutions*, *Web Services Journal*, volume 03 issue 06, June 2003.
7. Kornak and J. Distefano, *Cap Gemini Ernst & Young Guide to Enterprise Wireless Application Architecture*, Wiley, 2002.
8. J. Kubiataowicz, Extracting Guarantees from Chaos, *Communications of the ACM*, Volume 46, Number 2, February 2003.
9. D. Narayanan, J. Flinn, and M. Satyanarayanan, Using History to Improve Mobile Application Adaptation, *Proceedings of the Third Workshop on Mobile Computing Systems and Applications*, Monterey, CA, December 2000.
10. L. D. Paulson, *Computer System, Heal Thyself*, Computer, August 2002.
11. G. R. Ribeiro-Justo and A. Saleh, Non-functional Integration and Coordination of Distributed Component Services, *Six European Conference on Software maintenance and reengineering*, March 11-13, 2002, Budapest, Hungary, IEEE Computer Society.
12. G. R. Ribeiro-Justo and P. R. F Cunha, An Architectural Framework for Evolving Distributed Systems, *Journal Of Systems Architecture* 45, 1375–1384, 1999.
13. G. R. Ribeiro-Justo and T. Karran, An Object-Oriented Organic Architecture for Next Generation Intelligent Reconfigurable Mobile Networks, *3rd International Symposium on Distributed Objects and Applications, DOA 2001*, 17–20 September 2001, Rome, Italy, IEEE Computer Society.
14. SAS, *The SAS® Information Delivery Architecture: Introduction to Information Technology Manager*, 2001.
15. J. Schekkerman, *Adaptive Solutions: Concept Overview*, Cap Gemini Ernst Young, 2002, V.100.
16. F. Waelchi, The VSM and Ashby's Law as Illuminations of Historical Management Thought, in *The Viable System Model: Interpretations and Applications of Stafford Beer's VSM*, R. Espejo and R. Harnden editors, Wiley, 1996.

Towards a Semantic Integration of Medical Relational Databases by Using Ontologies: A Case Study

Tom Deray¹ and Pieter Verheyden²

¹ Language and Computing (L&C), N.V.,
Hazenakkerstraat 20A, B-9520 Zonnegem - Belgium,
Tom@landc.be

<http://www.landcglobal.com>

² Vrije Universiteit Brussel - STARLab
Pleinlaan 2, Gebouw G-10, B-1050 Brussel - Belgium,
pverheyd@vub.ac.be
<http://www.starlab.vub.ac.be>

Abstract. This report discusses a suitable and working solution for the semantic integration of dispersed, medical relational databases by coupling the databases to a medical ontology. We will demonstrate how this can be done by means of a case study, and how the coupling result can be deployed to query a relational database at the ontology level. Next to that, we will introduce a coupling language, and a discussion on how to integrate the language in two ontology models.

Keywords: Medical ontology, medical relational databases, database integration, semantic integration, ontology tools, coupling language

1 Introduction

In the health care sector, access to correct information in an efficient time frame is a necessity. Health care organizations (e.g., hospitals) may have several dispersed data sources containing specific information, though interrelated in some way. These data sources may vary from totally unstructured text (e.g., patient discharge notes written in some natural language by a medical doctor) to highly structured repositories (e.g., relational databases). A real-life example is a hospital consisting of several divisions. A central repository would contain administrative information of patients registered at the hospital, while every division possibly contains additional repositories of specialized information about treated patients. Since information about a patient located in a particular division's repository may be relevant to other divisions as well, a framework allowing the querying of those sources in a unified way is highly desired.

The solution currently proposed for this problem, is the *syntactic coupling* of different relational databases based on their structure, and the exchange of information through *messaging*. However, this is just a partial solution since

no use is made of the semantics, i.e. the meaning of the information. A more appropriate solution would be the *semantic coupling* (i.e. on knowledge level) of those relational databases to an ontology or formal description of a domain, in this case the medical domain of a hospital.

Language and Computing (L&C) and VUB STARLab join hands in an effort to find a suitable solution for this problem in the SCOP (Semantic Connection of Ontologies to Patient data) IWT-Eureka project. The collaboration between the industrial and academic partner is mainly focused towards the integration of dispersed relational databases using ontologies as central conceptual schemata.

VUB STARLab is concerned with the development of a coupling description language, called Ω -RIDL (Ontology Reference and IDEa Language), able to define a coupling between a relational database and an ontology in a natural manner. A guiding, step-by-step methodology for using this language to specify such a coupling will also be developed.

L&C wants to extend its ontology management server (OMS) LinKFactory® [Ceusters et al., 2001] with a component able to integrate information from external relational databases, and to offer a working solution for coupling relational databases to their LinKBase® ontology [Flett et al., 2002]. The LinKBase® ontology is a very large medical ontology, containing over 2,000,000 medical concepts that are interrelated by over 5,300,000 relations. Both concepts and relations are language-independent, but are cross-references to about 3,000,000 terms in various languages¹.

2 The MaDBoKS System

LinKFactory® is an ontology management tool designed to build, manage, and maintain large and complex language-independent ontologies. The LinKFactory® ontology client-server structure is an information system (IS) constructed using a typical 3-tier architecture (figure 1):

- LinKFactory® Workbench: client application on the user's computer system (or any other application using the LinKFactory® server functionality) to manage the LinKBase® ontology (client tier),
- LinKFactory®: the server interface, receiving and answering user requests, holding the business logic and requesting data using the data-server tier (application-server tier),
- The data layer accessing the underlying database. This database contains all information the LinKFactory® server needs to operate (user information, ontology content and maintenance information) (data-server tier).

As a preliminary step, L&C already developed the MaDBoKS (Mapping DataBases Onto Knowledge Systems) system as an extension to the

¹ At the time of writing, 16 languages are supported with primary focus on the following 7 languages: English (2,000,000 terms), Dutch (330,000 terms), Italian (141,000 terms), French (112,000 terms), Spanish (83,000 terms), Turkish (76,000 terms) and German (62,000 terms).

LinKFactory® OMS. MaDBoKS allows mapping of schemata and content of relational databases to LinKBase®² and querying of those relational databases on ontological level. The MaDBoKS system is developed as a working solution to some fundamental problems in a conceptual knowledge modelling system. A first problem concerns the modelling of so-called instance data. An ontology contains concepts that can have multiple instances (e.g., there is only one concept *human*, but there exist billions of humans in the world). In most cases, it is unfeasible both technologically and practically to model all instances in the ontology. The MaDBoKS system offers a solution by keeping instance data in external databases, and linking those databases to the OMS. A second problem concerns the redundancy and dispersion of data. It is possible in a company that several databases are dispersed over the company, without any physical link but with a *logical* link. Some information in one database could be merged with information in a second database, or some data could be redundant over more than one database, meaning that the different databases contains different kinds of information about the same data. By using the MaDBoKS system, it is possible to map both databases to the OMS, which would make the OMS a single point of entry to view all (mapped) relationships some information may have over several databases.

The coarse architecture of LinKFactory® and MaDBoKS is presented by figure 1. MaDBoKS uses a common architecture (used in the integration of different heterogeneous data sources) to implement the access to these heterogeneous databases.

MaDBoKS extends the data access layer of the LinKFactory. This way, it is abstracted towards the business logic, which can operate on the data without having to worry about its form or origin.

The different external databases are wrapped using wrappers. These wrappers offer a unified access towards the underlying information sources. The wrappers are responsible for resolving minor incompatibilities between the query languages used by different database vendors³.

Mediators [Wiederhold, 1992] transform the ontological requests into queries that can be processed by the wrappers. The data retrieved by those queries is then transformed into ontological information and returned to the data access layer. Every mediator uses an XML mapping file that defines the precise mapping between the external database schema (including population) and the ontology to generate the necessary queries to send to the underlying database wrapper.

The proxy module of the MaDBoKS receives all requests made to the data access layer. It manages these requests and sends them to the mediators if deemed necessary. It can check if information in the external databases is relevant to answer the requests by questioning the mediators (since the proxy module does not have specific knowledge of the mappings between the external databases and the ontology).

² Or any other ontology in use by the LinKFactory® server.

³ Most vendors use SQL with some minor modifications.

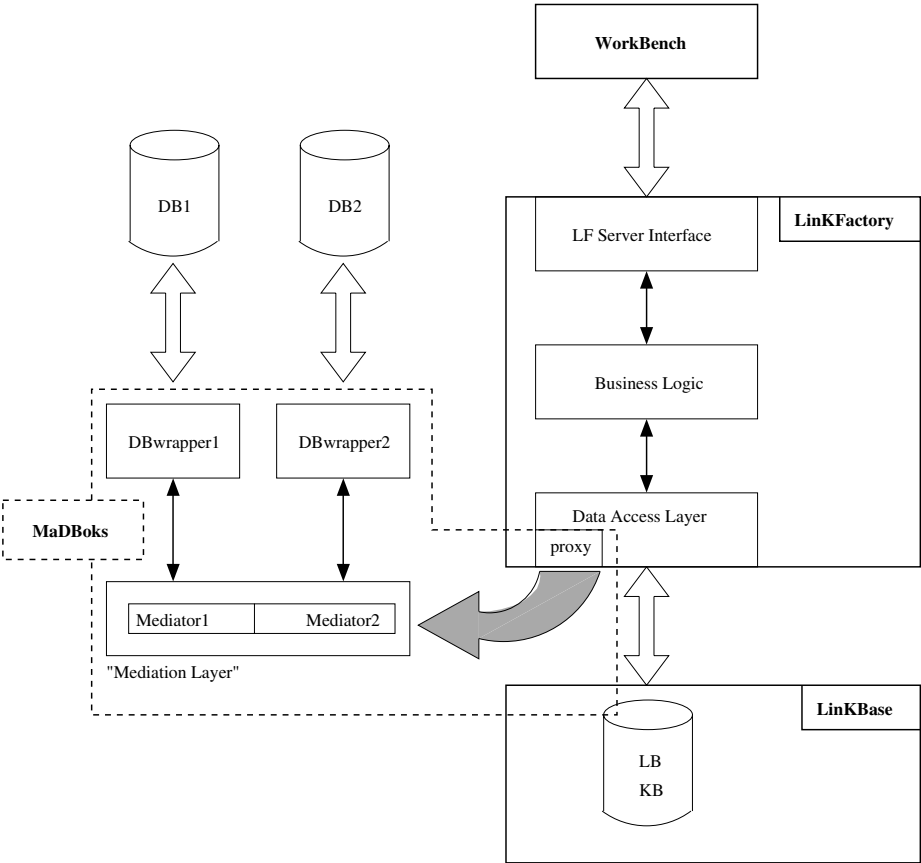


Fig. 1. A MaDBoKS system architecture.

We will briefly demonstrate how the MaDBoKS system is used to map a relational database onto the LinkBase® ontology by using the National Drug Code directory (NDC) of the Food and Drug Administration of the United States (FDA) as a case study. The NDC system serves as a universal product identifier for human drugs, and contains all kinds of information (e.g., ingredients, route of administration, package form, dosage form, type of drug, etc.)⁴.

As stated above, a MaDBoKS mediator uses an *XML* file to define the mapping between the relational database and the ontology. We can thus distinguish two phases in the process of mapping and deploying a relational database onto an OMS using MaDBoKS:

⁴ More information about the NDC system, and its freely available data files, can be found on <http://www.fda.gov/cder/ndc/index.htm>

- A **first** phase where the schema and population of the relational database is analyzed and mapped to the ontology semi-automatically. The result of this process is the creation of an (XML) mapping definition file.
- A **second** phase where the relational database gets physically linked to the OMS. This phase allows users to query the ontology with the integrated databases. Therefore, the OMS needs a mapping file to translate database information automatically to the ontology and vice versa.

In the **first phase**, the database is analyzed in a semi-automatic way. Tables and columns (and possible relations between them) from the database are conceptually modelled and compared to existing relations and concepts in the OMS. The workbench client offers the user a graphical environment to map the database to the ontology and create a mapping file. In this environment, several tools assist the user during the process. However, the final decision of mapping a specific database item to an ontological entity remains the user's responsibility, and is never done automatically by the system. An example of mapping a database column on an ontological concept is presented by figure 2. The column `ROUTE_NAME` of the table `ROUTE_OF_ADMINISTRATION` is mapped on the ontological concept `ROUTE OF ADMINISTRATION`. Basically, if we map a column on an ontological concept, the individual records of that column are becoming *instances* of that concept. The MaDBoKS system can perform some syntactic transformations on the table and column name to try and fetch a possible matching concept on which the column can be mapped. In this case, a correct match was not difficult to derive from the name of the column, so the user agreed with the result and assigns the mapping. Similar mappings have to be specified for as much columns as possible.

Unfortunately, there are obstacles making it difficult for the system to map concepts fully automatically:

- Table and column names can be cryptical, due to the use of non-standard abbreviations.
- In a large ontology, a specific name may map on several concepts.
- In a small ontology, a specific name may not map on any of the concepts.

Besides mapping on meta-data level, the MaDBoKS system also allows mapping on record level. This is necessary because not all population of a database is said to be only instance information. For example, the column `ROUTE_NAME` contains a record `BUCCAL` that can be mapped on the ontological concept `BUCCAL ADMINISTRATION ROUTE` which is a sub-concept of the concept `ROUTE OF ADMINISTRATION`. Because individual records can reflect ontological knowledge as well, the records of the database must therefore also be analyzed and mapped if necessary. Again, the system offers functionality to search for matching concepts on which a specific record can be mapped. However, the decision of mapping a particular record to a specific concept remains the user's responsibility, the system can only propose possible mappings. The result of the first phase (i.e., the definition of the mapping) is saved server-side as an XML file.

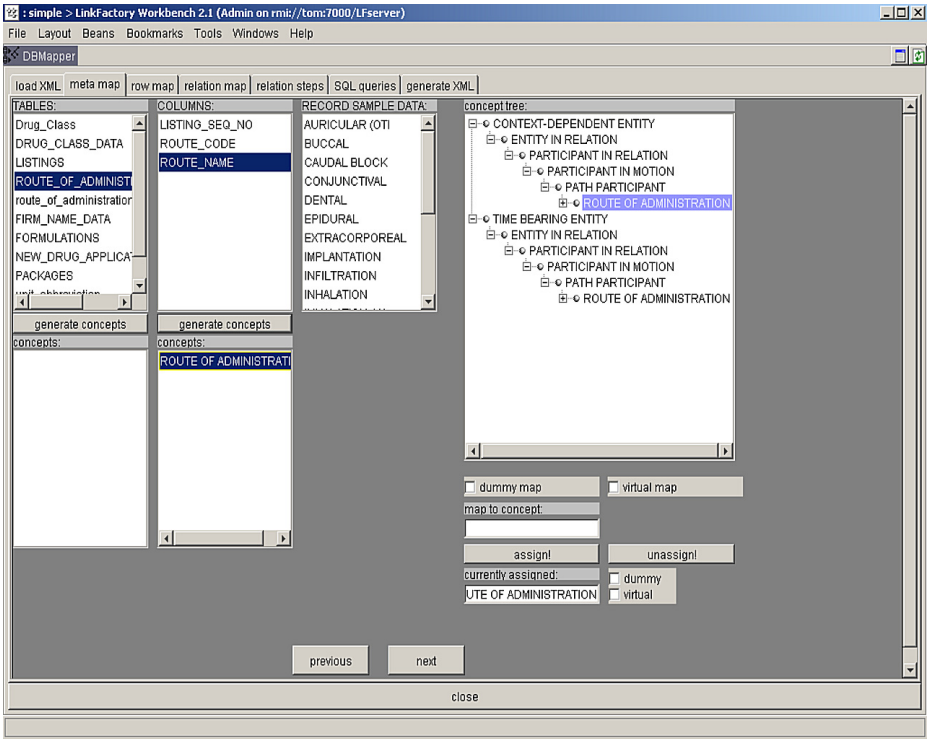


Fig. 2. A screenshot of the LinKFactory® Workbench with the MaDBoKS system showing an example of mapping a database column on an ontological concept.

In the **second phase**, the MaDBoKS system can use the XML mappings, defined in the first phase, to map relational databases dynamically on the LinkBase® ontology. Figure 3 presents the results of deploying the NDC mapping on the LinkBase® ontology. The content of the NDC database can now be browsed by the workbench client (or any other application accessing the LinKFactory® interface) as if it was part of the ontology. This means that all relations mapped on the ontology can be localized in the ontology. As an example, we searched for a concept beginning with 'OXY IR', and the system found the concept OXY IR CAPSULES from the NDC database. All the relations and reverse relations this concept has with other concepts (originated from the ontology or the database) have been retrieved *automatically* from the NDC database using the mapping information defined in the first phase. Each application using the ontology can now use the mapped content of the NDC database as well. This is very convenient for reasoning tools using the ontology as domain model, since they can now inference over the NDC information.

We have now presented a solution for the semantic integration of dispersed relational databases by coupling the databases to an ontology. We have shown

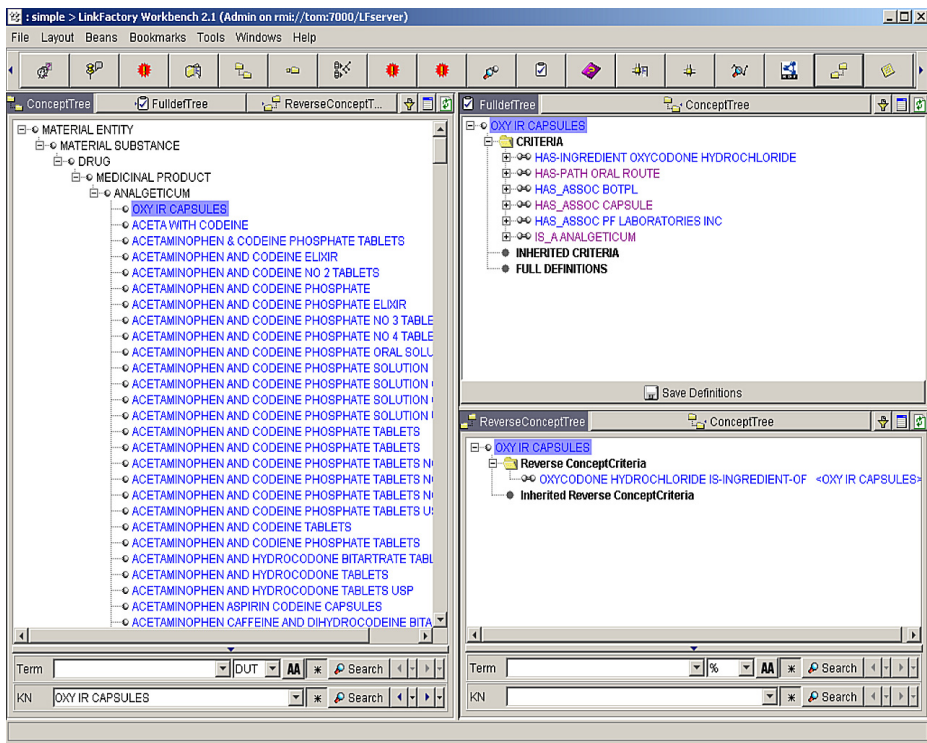


Fig. 3. A screenshot of the LinkFactory® Workbench showing the results of deploying the NDC mapping on the LinkBase® ontology.

how the integrated information can be visualized and browsed through the workbench client (or any other application using accessing the OMS interface). In the next section, a small example will be given on how we can use the OMS to specify more advanced (nested) queries on the ontology and the integrated sources.

3 Querying at the Ontology Level

Linking relational databases to an ontology must serve a purpose. The first, direct consequence is that by linking more than one relational database, we can integrate dispersed data sources⁵. Already, advantage is drawn from the mere fact that the LinkFactory®/MaDBoKS system offers access to information stored in the integrated data sources. A second consequence is the semantic meaning that the information from the relational database gains. This allows us

⁵ MaDBoKS allows the deployment of more than one relational database at the same time.

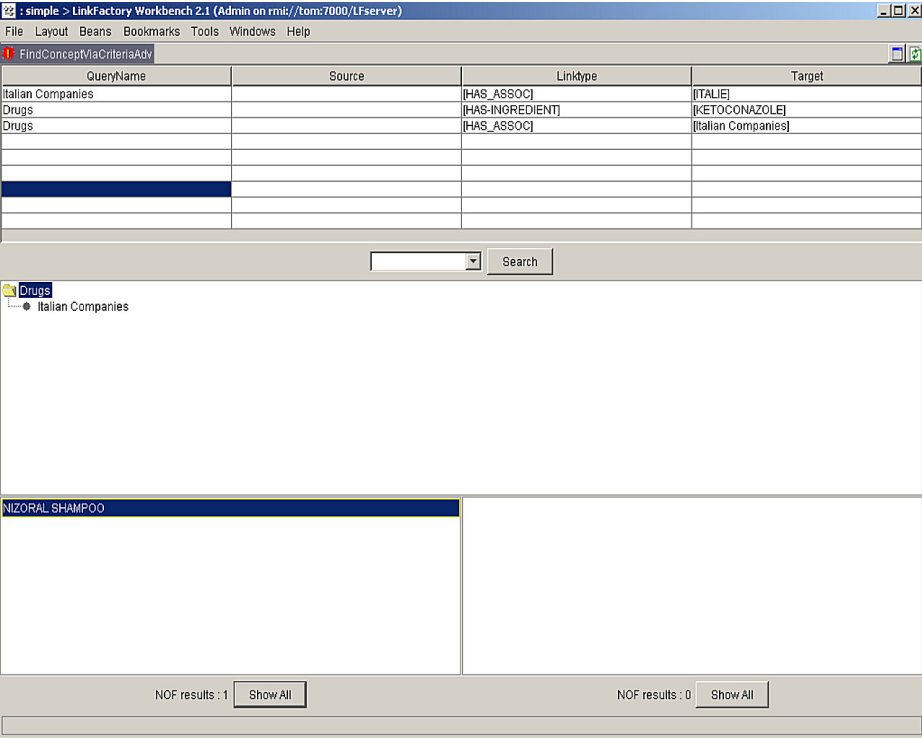


Fig. 4. A screenshot of the LinkFactory® Workbench showing the extension that enables querying at the ontology level.

to use any inference engine that works with the ontology, on the database data as well. Again, we will demonstrate this by means of a small example.

L&C developed an extension for the LinkFactory® that enables querying at the ontology level. When linking an external relational database with MaDBoKS, the information from that database can be queried at the ontology level. Figure 4 presents this extension. As an example, we would like to find all drugs containing the ingredient 'ketoconazole' produced by Italian companies. We set up the query by creating a first sub-query (named 'Italian Companies') that will fetch any company associated with Italy. Because only companies are associated with the concept ITALIE, we do not have to add a second constraint stating that the sub-query is a company. If country concepts had more associations, we would have to add another line named 'Italian Companies' with relation IS_A and as target concept ENTERPRISE. The results of the first sub-query are used by the second and final sub-query (named 'Drugs'). As a first constraint, we state that the potential 'Drugs' query results must have 'ketoconazole' as ingredient. The second constraint for the query states that the results must have an association with the results of the 'Italian Companies' query. Again, since we know that

only drugs have ingredients and only drugs are associated with companies, we do not have to state that the results must be drugs (although specifying this constraint does not alter the results). Through the nesting of these queries, we can now query information from the ontology and the external databases. Our small example returns the concept NIZORAL SHAMPOO, which is an anti-fungal and a topical anti-infective.

4 Extending DOGMA with Ω -RIDL

DOGMA⁶ is a research initiative of VUB STARLab where various theories, methods and tools for ontologies are studied and developed. A DOGMA inspired ontology is based on a model-theoretic perspective [Reiter, 1984] which decomposes an ontology into an *ontology base* and a layer of *ontological commitments*. This is called the principle of *double articulation*. An ontology base holds (multiple) intuitive conceptualization(s) of a domain (e.g., medicine). Each conceptualization is simplified to a set of context-specific binary fact types (relationships between concepts) called *lexons*. A lexon is formally described as $\langle \gamma \text{ term}_1 \text{ role co-role term}_2 \rangle$, where γ is an abstract context identifier used to group lexons that are logically related to each other in the conceptualization of the domain. The layer of ontological commitments mediates between the ontology base and its applications. Each such ontological commitment defines a *partial semantic account of an intended conceptualization* [Guarino et al., 1995]. It consists of a finite set of domain rules that specify which lexons of the ontology base are visible (i.e., *committable*) for usage in that commitment, and rules that semantically constrain this view. Experience shows that it is much harder to reach an agreement on domain rules than one on conceptualization [Meersman, 2002]. More details on the DOGMA approach can be found in [Jarrar et al., 2002] and [Spyns et al., 2002].

Figure 5 presents the global architecture of the current DOGMA framework for ontology engineering, consisting of an ontology server and a modelling tool called DogmaModeler. The research prototype tool DogmaModeler offers functionalities for modelling, browsing, and managing both the ontology base and the ontological commitments [Jarrar et al., 2003]. It supports the modelling of ontological commitments using the ORM (Object Role Modelling) diagram notation, the conceptual modelling language of Halpin [Halpin, 2001] which we largely adopted to design ontological commitments because of its semantic richness. Currently, an ontological commitment is expressed in ORM-ML, an XML-based markup language we developed for textually representing ORM diagrams [Demey et al., 2002].

The purpose of developing Ω -RIDL is to extend and refine our current method of building and expressing ontological commitments in DOGMA. The main syntactic principles of Ω -RIDL are adopted from RIDL, an old conceptual query/update and description language developed by Control Data in 1979

⁶ Developing Ontology-Guided Mediation for Agents.

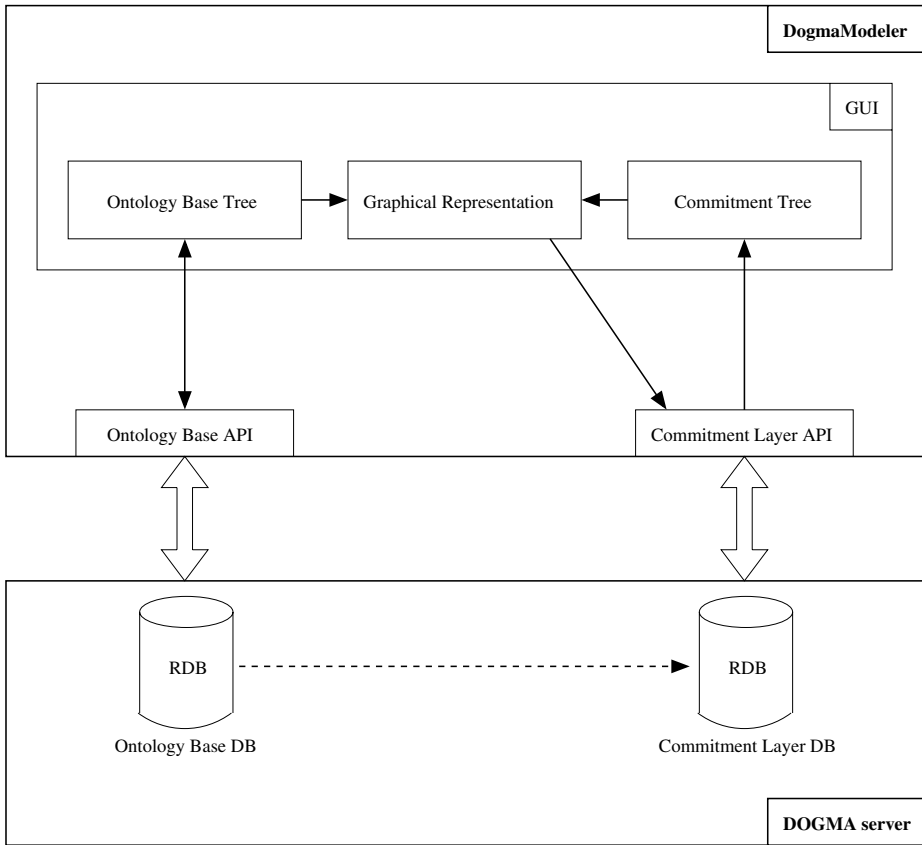


Fig. 5. Architecture of the DOGMA framework.

and used in the eighties and early nineties as a "conceptual interface" on top of a relational database [Meersman, 1982]. The conceptual and natural nature of RIDL was based on NIAM (Natural Information Analysis Method) which is the predecessor of ORM [Verheyden et al., 1982].

Currently, a prototype version of the Ω -RIDL language is formally specified, including a step-by-step methodology for using this language to describe an ontological commitment of a relational database to an ontology base. We will introduce the Ω -RIDL language with an example corresponding to a very small sample of the ontological commitment of the NDC database to the LinkBase® ontology which we converted to a DOGMA ontology base.

define commitment of NDC in context MEDICINE as

```
lexical-interpretations
  map FIRM_NAME_DATA.PO_BOX
  on POSTBUS ( [] HAS-ASSOC ENTERPRISE )
```



```

map FIRM_NAME_DATA.COUNTRY_NAME
on  'COUNTRY STATE' ( [] HAS-ASSOC ENTERPRISE )
end

lexical-associations
  assoc FIRM_NAME_DATA.FIRM_NAME with ENTERPRISE
end

semantic-constraints
  each ENTERPRISE HAS-ASSOC exactly one 'COUNTRY STATE'
  each ENTERPRISE HAS-ASSOC at-most one POSTBUS
end

end

```

An ontological commitment defined in Ω -RIDL consists of three layers. The *lexical interpretation layer* defines a mapping between the application vocabulary and the vocabulary defined by the conceptualization. In this case, lexical mappings are defined between database related symbols (e.g., attributes of certain relations) and ontological related concepts. Basically, we map a reference path from the application onto a reference path in the ontology base. In our example, all relational database reference paths are attributes that resides in a certain relation. The Ω -RIDL terminal symbols [and] are used to indicate that no explicit relation is modelled between two ontological concepts. For example, there exist a relation HAS-ASSOC between ENTERPRISE and POSTBUS, but there is no relation modelled between POSTBUS and ENTERPRISE. Why such a relation modelling decision in LinkBase® is made, will be explained in the following section.

The *lexical association layer* defines lexical associations between database symbols (e.g., attributes) that cannot be mapped on specific ontological concepts, but can be directly related (i.e., *associated*) with particular ontological concepts. In our example, the attribute FIRM_NAME from the relation FIRM_NAME_DATA is associated with the concept ENTERPRISE. This is because no ontological concept relationship ENTERPRISE HAS-ASSOC ENTERPRISE NAME is provided in the LinkBase® ontology.

The *semantic constraint layer* accounts for the intended meaning of the conceptualization. It is thus a set of axioms in terms of the visible lexons, that reflects the rules that are specified by the Universe of Discourse of the application (e.g., integrity constraints for a database). These constraints are designed in such a way that it models approximate as good as possible the intended models (e.g., intended database states) of the application language, according to the lexical interpretation. In general it's not so straightforward to find the right set of axioms, so that an ontology will admit other models as the intended ones [Guarino et al., 1995]. Therefore, Ω -RIDL provides a more expressive way for "specifying" the conceptualization such as mutual exclusion and subset constraints.

An implementation infrastructure to integrate Ω -RIDL in the DOGMA framework is planned in the near future.

5 Integrating Ω -RIDL into the MaDBoKS System

The MaDBoKS system would certainly benefit in supporting a formal coupling language like Ω -RIDL. Combined with a methodology for coupling a relational database to the ontology in a step-by-step manner using the Ω -RIDL specification, the MaDBoKS system could formally tighten its first mapping phase, offering a robust coupling formalism to use in the second mapping phase (i.e., the physical coupling of the external database to the ontology).

However, before an integration can be established, some obstacles will require more attention because the two ontology approaches (i.e., L&C's LinKBase® and VUB STARLab's DOGMA) show some differences that could influence the integration. One of these differences is how relations between concepts are perceived by both ontology models. Let us clarify this difference with a small example. In the real world, every infection is caused by bacteria. Therefore, we can model the following relationship in the ontology between the two corresponding concepts: INFECTION IS-CAUSED-BY BACTERIA. Although the reverse relationship (i.e., BACTERIA IS-CAUSE-OF INFECTION) is true for some cases, it is not true for all cases because some bacteria do not cause infections. Therefore, the ontology engineer is not allowed to model this reverse relationship in the LinKBase® ontology. In DOGMA, an ontology engineer is allowed to model each possible relationship between concepts in the ontology base. A constraint stating that some relation is mandatory, needs to be specified in the commitment layer. For example, a committing relational database containing information about all bacteria that do cause infections, will need to define a rule in its ontological commitment specifying that: each BACTERIA IS-CAUSE-OF at-least one INFECTION. Thus, each modelled relationship in a DOGMA ontology base is constraint-less, where each modelled relationship in the LinKBase® ontology is predefined as being mandatory.

As a result, the slight differences in interpretation between both ontology models will have to be carefully examined before applying the Ω -RIDL methodology to the MaDBoKS system.

6 Conclusion and Future Work

In this report we have discussed how a semantic integration of dispersed, medical relational databases can be achieved by coupling these databases to a medical ontology. We described the MaDBoKS system which makes it possible to map and link medical relational databases to the medical ontology LinKBase®, and demonstrated how such a mapping can be deployed to query a relational database at the ontology level. We also introduced Ω -RIDL, a coupling description language in which the coupling of a relational database to an ontology can be defined.

Future work will mainly be focussed on the further development and refinement of Ω -RIDL, its methodology, and how the language can be integrated in both ontology models.

Acknowledgments. The work and research presented in this report are carried out in the context of the SCOP project (IWT O&O #020020/L&C), funded by the IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders). We also want to thank Pieter De Leenheer for providing valuable feedback on an earlier version of this report.

References

- [Ceusters et al., 2001] Ceusters W., Martens P., Dhaen C., and Terzic B., (2001), LinkFactory® : an Advanced Formal Ontology Management System, Victoria, Canada, October 2001, K-CAP 2001.
- [Demey et al., 2002] Demey J., Jarrar M., and Meersman R., (2002), A Conceptual Markup Language that supports interoperability between Business Rule modeling systems, in *Meersman R., and Tari Z. (eds.), On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings, LNCS 2519, pp. 19–35, Springer Verlag.*
- [Jarrar et al., 2002] Jarrar M., and Meersman R., (2002), Formal Ontology Engineering in the DOGMA Approach, *On the Move of to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings, LNCS 2519, Springer Verlag, pp. 1238–1254, Meersman R., Tari Z. et al. (eds).*
- [Jarrar et al., 2003] Jarrar M., Demey J., and Meersman R., (2003), On Using Conceptual Data Modeling for Ontology Engineering, in *Aberer K., March S., and Spaccapietra A. (eds), Journal on Data Semantics, Special issue on "Best papers from the ER/ODBASE/COOPIS 2002 Conferences", Vol. 11, Springer.*
- [Flett et al., 2002] Flett A., Casella dos Santos M., and Ceusters W., (2002), Some Ontology Engineering Processes and their Supporting Technologies, Siguença, Spain, October 2002, EKAW2002.
- [Guarino et al., 1995] Guarino N., and Giarretta P., (1995), Ontologies and Knowledge Bases: Towards a Terminological Clarification, in *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, N. Mars (ed.), IOS Press, Amsterdam, pp 25–32.*
- [Halpin, 2001] Halpin T., (2001), Information Modeling and Relational Databases (From Conceptual Analysis to Logical Design), Morgan Kauffman.
- [Meersman, 1982] Meersman R., (1982), The High Level End User, in *Data Base: The 2nd Generation, Infotech State of the Art*, Vol. 10, No. 7, Pergamonn Press, U.K.
- [Meersman, 2002] Meersman R., (2002), Semantic Web and Ontologies: Playtime or Business at the Last Frontier in Computing?, in *NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises, pp. 61–67.*
- [Reiter, 1984] Reiter R., (1984), Towards a Logical Reconstruction of Relational Database Theory, in *Brodie M., Mylopoulos J., Schmidt J. (eds.), On Conceptual Modelling, Springer-Verlag, pp. 191–233.*

- [Spyns et al., 2002] Spyns P., Meersman R., and Jarrar M., (2002), Data Modelling versus Ontology Engineering, *SIGMOD Record Special Issue on Semantic Web, Database Management and Information Systems 2000* 31(4):12–17, Sheth A., and Meersman R. (eds).
- [Verheyen et al., 1982] Verheyen G., and Van Bekkum J., (1982), NIAM, aN Information Analysis Method, in Olle T., Sol H., and Verrijn-Stuart A. (eds), *IFIP TC-8 Conference on Comparative Review of Information System Methodologies (CRIS-1)*, North-Holland.
- [Wiederhold, 1992] Wiederhold G., (1992), Mediators in the architecture of future information systems, *IEEE Computer*, 25:38–49.

HCISWWA 2003 PC Co-chairs' Message

August 2003 Tharam S. Dillon, University of Technology Sydney, Australia
Elizabeth Chang, Curtin University of Technology, Australia
(Workshop on Human Computer Interface for Semantic Web
and Web Applications 2003 Program Committee Co-chairs)

Designing Ontology-Based Interactive Information Retrieval Interfaces

Elena García¹ and Miguel-Ángel Sicilia²

¹ Computer Science Department, Alcalá University,
Ctra. Barcelona km. 33600 – 28871 Alcalá de Henares (Madrid), Spain
elena.garciab@uah.es

² Computer Science Department, Carlos III University,
Av. Universidad, 30 – 28911 Leganés (Madrid), Spain
msicilia@inf.uc3m.es

Abstract. The so-called Semantic Web advocates the future availability of machine-understandable metadata, describing Web resources by means of ontologies expressed in description logics. This would eventually entail changes in Information Retrieval (IR) indexing and matching algorithms, but also in the user interface design of IR tools. This second aspect can be informed by existing Interactive Information Retrieval (IIR) research, but it requires also further investigations about the interaction of users with terminological structures and iterative, browsing-oriented query construction paradigms. In this paper, preliminary experiences and reflections regarding ontology-based query formulation interface design are described.

1 Introduction

Information Retrieval (IR) can be defined as a discipline concerned with the formulation and resolution of queries issued by users against a database of information items (possibly heterogeneous in format and structure). According to [3], the effective retrieval of relevant information is affected both by the *user task* and the *logical view* of the information items. The interaction of the user with the IR system usually comprises both retrieval and browsing activities, oriented toward fulfilling an information need. The logical representation of an item is a representation of its contents, in many cases consisting on a set of keywords extracted by humans or by means of automated mechanisms. Both elements have been extensively studied in the last decades from two complementary perspectives: a system approach and an interaction approach. The system approach is mainly concerned with the process of matching a query against the database of items, in an attempt to produce optimal rankings. The interaction approach — often referred to as Interactive Information Retrieval (IIR) — puts emphasis in human factors, stressing the iterative nature of information searching (a review can be found in [20]), and extending the scope of IR to the needs, motivations and strategies of users in their interaction with IR systems.

Classical IR algorithms are founded on the assumption that information items (or documents) are modeled logically by keywords pertaining to some natural language. But the vision of a Semantic Web [7] radically changes this perspective. The Semantic Web essentially advocates “crossing the chasm” from unstructured keyword-based models to richer logic-based annotations that would eventually provide a basis for reasoning. This entails that the logical model of a document becomes a set of logical assertions about its contents (and perhaps also about its physical structure, its relationships with other documents and other information). In addition, the form of the queries becomes a logic expression with an arbitrary level of complexity in its structure. Both consequences, when taken together, lead to a reconsideration of existing IR user interfaces, beyond the mere typing of search terms. The problem can be divided again in a system-oriented part and an interaction-oriented aspect. The former is mainly a matter of logical satisfaction and concerns annotation approaches, and eventually problems of logical approximation in the sense given in [17] and elaborated, for example, in [25]. The latter concerns the interaction strategies of the user with one or several linked terminological structures, and the interpretation of her actions as query-formulating criteria, comprising activities at various levels of granularity, that, according to [5], can be roughly categorized as moves, tactics, stratagems and strategies. In this paper, we mainly focus on this second interaction aspect, in an attempt to provide a point of departure for the design of a new generation of IR interfaces based on formal ontologies (it should be noted that formal ontologies are different to lexical thesauri [27] in that they are free of some problems of ambiguity that occur in natural language) that provide an effective and efficient interaction relying on logics-based techniques like those described in [18]. Moreover, the results and design guidelines provided in this paper are based on previous research on a concrete ontology-driven IR system described in [22] and [13], and they are also informed by previous research on IIR in a broad sense. It should be noted that the criteria sketched in this paper do not exhaust the wide range of IR possibilities opened by Semantic Web technologies, and further research is needed both to accumulate experimental evidence (whenever semantic metadata becomes mainstream available) and to develop a methodological framework to construct and evaluate such systems. In consequence, the ideas discussed here are mainly directed toward stimulating further research.

The rest of this paper is structured as follows. In Section 2, the context and requirements of the problem is delimited, providing a set of assumptions about the evolution and the eventual deployment of the Semantic Web as a global infrastructure, just as the Web is today. From that initial context, a number of query formulation issues (and their related search strategies) are discussed in Section 3. Section 4 summarizes and discusses preliminary findings. Finally, conclusions and future research directions are provided in Section 5.

2 Some Assumptions Regarding Interactive Ontology-Based Search

Since the Semantic Web is an evolving and growing reality, any attempt to investigate one of its applications should first state the assumptions in which it's grounded. In consequence, we first provide a reasonable set of assumptions that conform the point of departure of our ontology-based IR research. In addition, a number of design requirements are established as tentative goals elaborated from existing research.

The first assumption states simply that the Semantic Web relies in description logics (DL) as its foundational technical substrate. Currently, this appears to be largely uncontroversial given the increase of research results in this direction¹, and provided that the OWL language, endorsed by the W3C consortium as a Semantic Web standard [12], includes a description logic sub-language.

Assumption 1 *The $\mathcal{ALC}$ description logic (or some of its extensions) will be used as the language(s) both for metadata annotation and to describe their associated ontologies.*

We mention $\mathcal{ALC}$ as a minimum, since it's a basic, significant representative of DLs. Of course, more expressive DLs are currently used (and can be expected to be used) in Semantic Web languages and prototypes.

Assumption 2 *Shared ontologies expressed in DL by means of standard formats will be available to IR systems.*

Although it may take a long time to have available a set of significant consensual ontologies covering most domains of everyday's information seeking, it must still be considered a prerequisite for Semantic Web applications, specially for general-purpose Web search engines. The emergence and availability of large conceptualizations like OpenCyc² represent an important step in that direction. Both assumptions (1) and (2) are in accordance to existing analysis regarding the technological basis of the Semantic Web, as recently stated by Horrocks et al. [14], since formal models are obviously better suited to machine understandability, and common conceptualizations are required to enable interoperability.

Assumption 3 *Web resources annotated through DL assertions regarding shared ontologies will be available as the item base and logical item model for IR systems.*

Assumption 3 entails that a ontology-based metadata record will be provided for *every* searchable Web resource. Of course, and given the growing size of the Web, this provision may be expected to become a reality in a gradual way, giving room to hybrid models of IR. Nonetheless, here we'll approach the problem

¹ <http://dl.kr.org/>

² <http://www.opencyc.org/>

from an ideal perspective in which metadata exists for the whole database of resources. Moreover, annotation is still an open problem from the viewpoint of the quality and consistency of metadata records, and currently information extraction techniques appear to be a good compromise semi-automated solution to annotation [10]. In any case, we'll assume here that annotations are properly constructed, just as conventional IR engines rely in their keyword-based logical models.

Assumption 4 *The user population consists of people who do not specialize in search and are who have not knowledge about ontologies or knowledge models.*

Assumption 4 states that the query formulation and resolution mechanisms should be devised to reach the vast majority of the profiles that currently can be found in the Web. This precludes designs based on specialized query languages, and also user interfaces that include technical elements that are part of the solution domain. An example of such design is the **Ontobroker** query interface as described in [11], in which the technical ontology concepts of object, class, attribute and value are used in 'combo-boxes' to form logical query expressions.

Some specific design requirements must now be added to the just discussed assumptions to conform the space of design possibilities we're looking for.

Design Requirement 1 *Information retrieval interfaces must provide support to iterative query refinement and, as a supplementary function, also to serendipitous discovery.*

One possible approach to design an ontology-based IR interface is that of simply adopting the existing layout of current search engines, in which the user types some words or phrases, and then browses the list of results. This is the approach taken by querying systems like **OntoQuery** [2], that still relies on natural language processing for query resolution. But here we focus on IR interface designs in which the query formulation process is interactive and more user-controlled. These kind of alternative, more sophisticated UI designs have been advocated by Bates [5], resulting in a classification of user involvement levels and associated search tactics, stratagems and strategies. The topic of discovery by chance has been raised in recent studies [26] as a way to enhance the overall usefulness of information seeking behaviors. Despite the available evidence about its value, it still remains to be investigated the concrete design tactics and the limits of this feature. In consequence, we'll restrict ourselves here to provide some hints about potential ways in which ontology-driven search may trigger serendipity encounters.

Design Requirement 2 *The design and affordances of the interface must be independent of the domain.*

Requirement 2 entails that we are concerned with general purpose designs. Some existing ontology-based search tools provide interfaces specific to a given domain. For example, **Melisa** [1] provides a Windows-Icon-Menu-Pointer

(WIMP) interface that gives the user the possibility to indicate ‘evidence quality’ and ‘integration of the evidence’ that are specific to medical or related domains. It should be noted that we don’t claim that domain-specific elements should be absent from the interface, we only commit to plug-ability of ontologies in the same runtime of the IR system, so that for all the domain-specific query formulation information must be encoded associated to the ontology, in a language understandable for the IR system.

Design Requirement 3 *The user interface must be Web-browser based.*

Web browsers are the dominant interaction paradigm in the current Web, and their particular, uniform hypertext navigation mechanisms will likely stay largely unaffected while the HTTP protocol remain stable. Consequently, Web usability and information architecture [21] guidelines are still relevant to Semantic Web-enabled interfaces. Additionally, our present research is restricted to human information seeking. Further studies should address the possibility of building information-seeking software agents and also collaborative information seeking strategies.

3 Query Formulation and Search Strategy Design

In surface, the browsing of the concept network determined by ontologies appears as analogous to navigation on existing link catalogs like *Yahoo!* [15]. But although the clickstreams of both designs may appear as similar, the cognitive intention of user actions would eventually be fairly different, and the result of the user task is not the same. In the latter case, information items are found by simply browsing nested subject categories lacking a formal taxonomic criterion, while in the former, relationships between terms and also subsumed categories can be used to specify an underlying query that may be resolved following diverse heuristic approaches. Thus, the ontology-driven interface designer is faced with both usability and technical issues that open a number of currently unexplored alternatives.

In this section, we describe the rationale for the main design problems we faced in the design of the prototype ontology-based IR system described in [22] and [13], that will be referred to as *OntoIR* from here on. Technically, the system works by querying a *Rdf*-based ontological model comprised by three related sub-ontologies and implemented on the *Tomcat* Java-based Web server³ using the *Jena Rdf*-processing libraries⁴. The three sub-ontologies describe respectively the *domain*, the *Web resources* annotated with concepts in that domain, and the *bibliographic sources* in which the domain concepts and relations are described. Any ontology serialized in the DAML+OIL language can be plugged into the tool, thus satisfying Design Requirement #2. Evaluation and user comments about *OntoIR* system work have resulted in a number of identified improvement areas,

³ <http://jakarta.apache.org/tomcat/>

⁴ <http://www.hpl.hp.com/semweb/jena.htm>

so that other design alternatives that were discarded or previously neglected are also suggested here, with the intention of motivating further research in the area.

The first problem encountered in the design of an ontology-based IR interface is simply where to start, that is, which elements of the ontology are provided as a guide for the user to begin the specification of the search.

Problem 1 *What are the elements that must be provided as a point of departure for the user task?*

In DL ontologies we have three basic candidates: concepts, relations between concepts, and their instances. If a search process begins from scratch, the problem of deciding which elements to show is limited by the practical design rule of limiting the length of pages [16] (thus avoiding scrolling). The number of concepts or terms in a given ontology may in most cases be in the range between a dozen and several hundred terms, while the number of objects is typically larger — a factor of the number of concepts. Since relationships may difficult the comprehension of the initial user move, it appears reasonable to begin with ontology concepts. Nonetheless, the use of relationships may be subject to future studies.

OntoIR is based in a top-down approach to query refinement, organized around concepts. According to it, the user first selects a domain from which to start the query, and then, the interface provides him/her with a number of what we call *entry points*, along with its descriptions. Entry points are a number of terms that are marked (with a form of meta-metadata) as query initiators, or that are selected from the profile of previous queries of the users, in the case of having an adaptive approach. The results and the terms that are provided by the system in a given search process are subject to become initiators for new searches, as discussed later. Thus, the information seeking problem is broken up in two (possibly iterative) steps:

- **Domain selection.** First, the user selects a topic (i.e. a domain or ontology) from which to start the search. It should be noted that this does not entail that the entire search will be limited to that ontology, since relationship traversal may lead to a query using terms from other domains, or combining terms from different ontologies. Since this step is not strictly related to query formulation, we'll not go into details about it. Given that in the future, the number of available ontologies may be relatively large (in accordance with Assumption #2), some form of hierarchical browsing — or even a classic IR process — should be provided for this selection step.
- **Query formulation and resolution.** The entry points of the selected ontology are provided as a point of departure. Therefore, it's advisable that the number of entry points be small enough to fit into the screen (or at least to minimize scrolling). The current UI design is shown in Figure 1.

If the system is provided with user modeling and adaptive capabilities, some form of *search-by-example* may be devised, taking previous queries or results of the user, or even from similar users if some form of *collaborative filtering* approach (see for example [19]) were included in the search interface.

Once in the query formulation step, a second range of alternatives arise.

Problem 2 *What are the user moves that contribute to query formulation and how do they contribute?*

Problem #2 refers to the interpretation of user interactions. The overall problem may be stated as how to translate a variable number k of discrete, sequential user moves (belonging to a set M) or interactions into a DL-based form. We have reduced the problem to the simpler one of collecting a set of terms $\mathcal{C}$ from user interactions as expressed in (1).

$$t : M^{(k)} \mapsto \mathcal{C} \quad (1)$$

The rationale for such simplification is that studies on current search practices like [24] have shown that queries are short (most of them including less than ten terms) and also simple in structure. Nonetheless, it should be noted that the transformation expressed in (1) loses the ordering of the selection of terms and their selection context, and precludes selecting relations, so that further research should generalize this model.

The screenshot shows the OntoIR tool interface. At the top, there is a menu bar with 'Archivo', 'Edición', 'Ver', 'Favoritos', 'Herramientas', and 'Ayuda'. Below the menu is a header section with the 'OntoIR' logo, the text 'Domain: Usability evaluation methods', a 'Select ontology' link, and a 'Search!' button. The main content area is divided into three sections: 'Concepts', 'Kind of Results', and 'Suggestions'.

Concepts section:

- ☐ **Inspection methods**
 - ☒ **Heuristic evaluation**: Technique where usability specialists judge whether each element follows established principles
 - ☐ **Cognitive walkthroughs**: Technique where experts construct task scenarios from a specification and then play the role of a user working with that interface
 - ☐ **Pluralistic walkthroughs**: Technique where users, developers, and usability professionals step through a task scenario, discussing each element of interaction
 - ☐ **Consistency inspection**: Technique to ensure consistency across multiple products from the same development effort
 - ☐ **Guideline inspection**: Technique to ensure that usability principles in guideline form are considered in the interface design
 - ☐ **Standard inspection**: Technique to ensure compliance with industry standards
- ☒ **User testing**
 - ☐ **Thinking aloud protocol**: Technique where participants are asked to vocalize their thoughts while interacting with the product
 - ☐ **Co-discovery method**: Technique where two participants attempt to perform tasks together while being observed
 - ☐ **Question-asking protocol**: Technique where you prompt the users by asking direct questions about the product
- ☐ **Artifacts**
 - ☒ **Questionnaires**: Written lists of questions that you distribute to your participants to collect information
 - ☒ **Guidelines**: Design principles to achieve usability
 - ☐ **Scenarios**: describes a sequence of events when interacting with a system from the users perspective

Kind of Results section:

- ☒ **Article**
 - ☐ **Journal article**: Contribution to a journal
 - ☐ **Article in book**: Chapter of a book
 - ☐ **Technical report**: Internal formal document of an organization or a company
 - ☐ **Conference paper**: Contribution to a conference

Suggestions section:

- User testing [Measure](#) Usability attributes [Move to concept 1](#) ↑
- Questionnaire [Measure](#) Satisfaction [Move to concept 1](#) ↑

Fig. 1. Overall appearance of the OntoIR tool

The current version of OntoIR allows for the following kinds of user moves that build the query in a top-down fashion:

- The “Concepts” table allows for the selection (via *checkboxes*) of terms. The ‘refine’ functionality allows the user to explore the next level of the hierarchy for the selected terms, till the leaves of the generalization-specialization network is reached. This means going from more general categories to more specific ones, enabling wider or narrower searches. One major alternative that was discarded in *OntoIR* was that of allowing the user to move from terms to their generalizations (i.e. bottom-up moves).
- The “Kind of Results” table allows for restricting the type(s) of Web resources to be retrieved, i.e. the form of the document. This is not limited to the format, but to the type of document. The (KA)² ontology [6] provides an example taxonomy of scholarly documents that has been adopted in *OntoIR*, but richer categorizations, perhaps coming from library science, may be expected in the future. The selection process follows the same iterative top-down approach used for the domain concepts.
- The “Suggestions” area provides a way to enrich the query formulation process with related concepts. In this area, concepts bearing relationships with elements in the ongoing term collection ($c' \in \mathcal{C}'$) are provided (i.e. domain concepts $d \in D$ such that $R(c', d)$ or $R(d, c')$ being R any domain relation). The ‘move to concepts’ functionality makes appear the concept involved in the given relationship in the “Concepts” area, that is, the related concept is added to the ongoing $\mathcal{C}'$. The terms shown in this area are randomly selected from the possible ones.

The second and third areas in the list above are “collapsible” and “expandable” to save screen space if required. In synthesis, query formulation proceeds top down, suggesting related terms, and providing an optional document type filter. The terms in the “Concepts” area are links that trigger a separate browser window showing the bibliographic source(s) from which the concept definition was borrowed. This is a useful user function to prevent concept misunderstandings.

Once the elements that conform the query have been collected, an interpretation step is required, stated in Problem #3.

Problem 3 *What is the approach taken to match the query with the contents of the item database?*

Now the problem may be stated as how to translate $\mathcal{C}$ into a DL-based expression denoting a number of Web resources. The overall form of this translation step may be denoted as $r : \mathcal{C} \rightarrow \Psi$, where Ψ is a concept expression denoting a number of Web resources (direct or indirect instances of a top class **Resource**). It should be noted that Ψ should not be substituted by a logically equivalent expression $E \equiv \Psi$, since the terms selected by the user carry a semantic content that may eventually determine the results of the query and also contribute to future analysis of search behaviors.

Other alternatives to mapping r may combine concept expressions with example instances and/or relation names, allowing for more flexible retrieval ap-

proaches. Expression (2) describes the query approach of **OntoIR** for this mapping.

$$r_{\text{OntoIR}} : \mathcal{C} \mapsto \overbrace{\mathcal{R}_{\mathcal{D}} \sqcap (T_1 \sqcup T_2 \dots \sqcup T_l)}^{\psi} \quad (2)$$

Expression (2) describes matching items as a concept expression $\mathcal{R}_{\mathcal{D}}$ restricted to the types of documents $T_1, \dots, T_l$ selected in the “Kind of Results” area (all types are allowed if no T_j was selected). $\mathcal{R}_{\mathcal{D}}$ is a combination of three sets of Web resources related with the sets of instances C, E and S . Expression (3) defines S in terms of the concepts $D_1, D_2, \dots, D_h$ selected in the “Concepts” interface area, as the concept covering all the instances of the domain concepts in $\mathcal{C}$, and C is defined as a subset of S in which an arbitrary domain relationship R exists to other element of S . In the same expression, E is defined as a set of concepts external to S that are related to one or more concepts in S .

$$C \doteq S \sqcap \exists R.S \text{ and } E \doteq \neg S \sqcap \exists R.S \text{ given that } S \doteq D_1 \sqcup D_2 \dots \sqcup D_h \quad (3)$$

Given that instances of **Resource** (i.e. Web information elements) are linked to elements in domain ontologies through relations (with concrete positive or negative meanings like **describes**, **criticizes** or other kind of relation semantics) subsumed by a top relation *ABOUT*, $\mathcal{R}_{\mathcal{D}}$ can be determined by expression (4).

$$\mathcal{R}_{\mathcal{D}} \doteq \text{Resource} \sqcap (\exists \text{ABOUT}.C \sqcup \exists \text{ABOUT}.E \sqcup \exists \text{ABOUT}.S) \quad (4)$$

Where *Resource* is the top class of all kind of Web resources in the ontology. From expression (4), relevance criteria for resources can be implemented regarding to each of the three sets C, E and S . **OntoIR** currently considers a simple preference relation $\exists \text{ABOUT}.C \succ S \sqcap \neg \exists \text{ABOUT}.C \succ \exists \text{ABOUT}.E$, but more elaborated approaches could be devised from the same setting. For example, the number of connections inside C may be considered as an indicator of relevance, and even flexible approaches similar to quantified statements in [8] may be approached.

An example of a page of query results is showed in Figure 2. These results come from the query formulated in Figure 1 on a *usability evaluation methods and techniques* domain. The query has been built to retrieve all kind of on-line articles about questionnaires and guidelines in user testing and heuristic evaluation methods. It should be advised that all these concepts are the result of a previous refinement of several entry points and some of them could be refined in turn (e.g. questionnaires). In the basis of a sample of annotated resources, **OntoIR** shows results according to the priority rule defined above. The (internal) set S of concept instances comprises **severity rating**, **QUIS 5.0**, **QUIS 6.0**, **SUS** and **OSF/Motif guidelines**, among others. The first instance represents an opinion questionnaire, the following three are satisfaction questionnaires and the last one is a set of guidelines. As the **Standard Inspection** technique is not selected and it represents an inspection method that uses, for example, **OSF/Motif**

guidelines, the generic **Standard Inspection** instance is included in set *E*. The set *C* contains instances like **severity rating**, **QUIS 5.0**, **QUIS 5.0** or the generic instance **heuristic evaluation**, since **QUIS 6.0** is a version of **QUIS 5.0** and **heuristic evaluation** uses **severity rating** questionnaires. For each retrieved document the following information is provided: (a) The kind of resource, (b) a brief extract of the content of the document, that allows the user to evaluate its suitability for his/her search, (c) some relevant citation information about the document, and (d) the relations it maintains with other instances of the selected terms. These relations may be to initiate new searches, since users can elaborate a new query with the terms involved in the relations via the “search using related concepts” functionality. In addition, the related instances are showed as links to access their description.

Obviously, interpretation requires further alternative explorations on realistic ontologies to have an idea of the appropriateness of such schemes. In addition, several alternative interpretations could be implemented, allowing the user of the system to decide which is better for the task at hand.

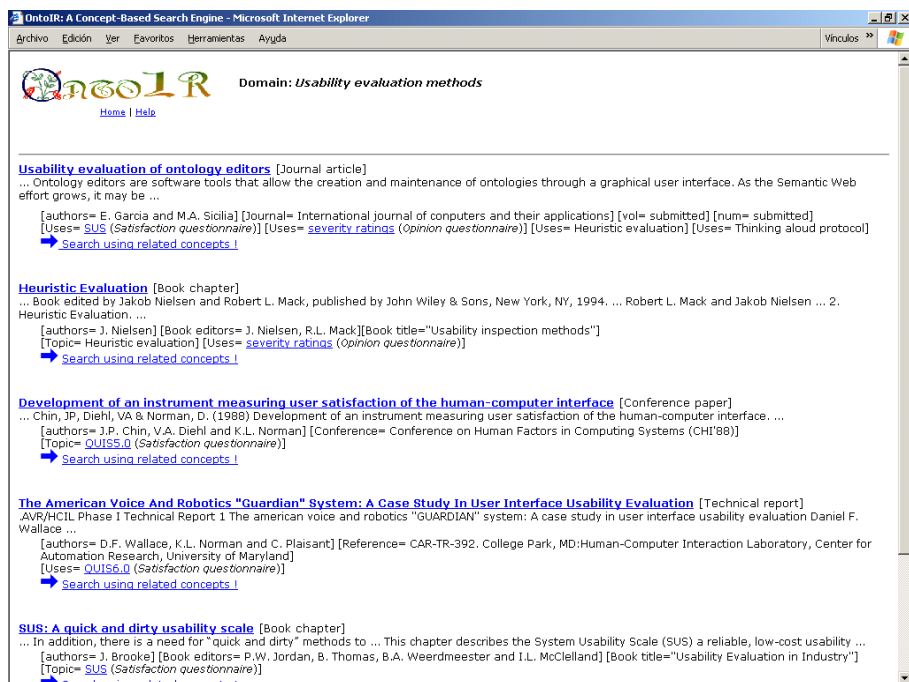


Fig. 2. A search results page in the OntoIR tool

The last problem that must be addressed is how to foster iterations and even casual encounters, as prescribed in Design Requirement #1.

Problem 4 *What mechanisms are provided to initiate a new search from a previous one?*

As we have advanced in the example, *OntoIR* provides a number of features that work as initiators of new search processes:

- Relationships suggested in the query formulation interface work as links leading to a separate search interface including the terms in the relation and also the entry points of their respective ontologies.
- Each of the search results is provided with the concepts related with it (along with the name of the relationship), and the “search using related concepts” functionality initiates a new search including the concept involved.

These affordances can be complemented by basic personalization features, including the history of past searches of the user, sorted by the number of times the user repeated it, and also a mode of search in which the user begins selection from the most selected concepts used by him/her in past searches. This is consistent with the considerable amount of repeated queries reported in [24].

In addition, some specific ontological axioms has been recently incorporated to *OntoIR* to experiment on casual searching behavior. Concretely, the *disjointWith* axiom of OWL has been used to provide query initiators based on concepts that are the opposite of those included in the ongoing query.

4 Some Preliminary Results and Reflections on Evaluation Techniques

According to the terminology proposed by Bates, our prototype system supports two levels of activity (falling into the “area of recommended development” proposed in [5]):

- At the “stratagem” level, the “Kind of Results” area has proved to be a separate mechanism for users to “Filter by Type” the ongoing search, in most cases carried out once at a time and before selecting domain terms.
- At the “tactic” level, the SUB (to move downward to a more specific term), RELATE (to move sideways to a coordinate term) and CONTRARY (to search for the term logically opposite) tactics [5] are provided. In addition, the SELECT tactic, i.e. breaking complex search queries into subproblems is somehow supported in the “Domain Selection” step, and the process of refinement — that only affects to selected terms — may be considered sometimes as a CUT tactic (to choose the option that eliminates the largest part of the search domain).

Although many search activities remain unsupported in our current prototype, its querying model is consistent to a large extent with evolving searching models like “berrypicking” [4], and the concatenation of search processes also

facilitates term relevance feedback [23], as preliminary evaluation has pointed out.

While we wait for the emergence of TREC⁵-like evaluation suites for ontology-based IR systems, IIR models of evaluation like [9] can be tailored to the specifics of the design space described in this paper. *OntoIR* has experienced two evaluation processes. The first one, reported in [13] led to the separation of the overall query formulation interface in three areas, and also to a reformulation of the r_{OntoIR} mapping. The second one is more recent and was approached as a user testing process with ten users using the thinking aloud protocol. The users were first introduced to the features of the tool, and then they were given an example query and several tasks consisting on concrete searches involving from three to six refinement moves. Two groups of users were formed: profile A included programming students (daily users of the Web), while group B was formed by non-technical people with casual experiences with conventional search engines. Groups A and B were given six concrete search tasks regarding the domain of sailing, group A was provided also with four tasks regarding an ontology of programming languages. Both ontologies (and their associated resource bases) were built specifically for evaluation purposes, containing about a hundred annotated resources. Failure rates (i.e. unfinished tasks) were of less than one in average, and the average time per search task was of about three minutes. No significant differences were found between groups A and B, with the exception of the results of an informal brief Likert-scale (one to five) questionnaire administered at the end of the sessions regarding ease of use, learnability and perception of usefulness (compared to conventional search engines). A difference of more than one point in the scale pointed out to a worse perception of ease of formulating queries in group B. Observations also pointed out to the appropriateness of including ways to move up in the hierarchy, and also to provide more clear differentiations of the “Refine” and “Search” moves in the interface. Other minor possible enhancements are providing a text describing the overall domain while formulating the query and also a way to indicate that no more sub-hierarchy levels are available for a given term.

5 Conclusions

The realization of the Semantic Web vision calls for a revised conception of classical IR user tasks and logical models. Since ontologies are considered a key service in the prospective Semantic Web [12], the design of ontology-based IR interfaces has become an important research issue in this area. In this paper, our first experiences in designing and evaluating such systems has been described, highlighting the main assumptions, design requirements and problems that are faced when approaching the problem from a human factors perspective. A prototype has been described that enables a number of search tactics in the context of top-down iterative query refinement. Further research is required to explore

⁵ TREC is the acronym for Text REetrieval Conferences:
<http://trec.nist.gov/pubs.html>

other design alternatives and also to develop standards and best practices regarding the evaluation of ontology-based IIR systems.

References

1. Abasolo, J.M., Gómez, M.: MELISA: An ontology-based agent for information retrieval in medicine. In: *Proceedings of the First International Workshop on the Semantic Web (SemWeb2000)*, Lisbon, Portugal (2000) 73–82
2. Andreasen, T., Fischer-Nilsson, J., Erdman-Thomsen, H.: Ontology-based Querying. In: Larsem, H.L. et al. (eds.) *Flexible Query Answering Systems, Flexible Query Answering Systems, Recent Advances*, Physica-Verlag, Springer (2000) 15–26
3. Baeza-Yates, R., Ribiero-Nieto, B.: *Modern Information Retrieval*. ACM Press, Addison-Wesley NY (1999)
4. Bates, M.J.: The Design of Browsing and Berrypicking Techniques for the Online Search Interface. *Online Review* **13** (1989): 407–424
5. Bates, M.J.: Where Should the Person Stop and the Information Search Interface Start? *Information Processing & Management* **26** (1990): 575–591
6. Benjamins, R., Fensel, D., Decker, S.: KA2: Building Ontologies for the Internet: A Midterm Report. *International Journal of Human Computer Studies*, 51(3) (1999) 687–713
7. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American*, **284**(5) (2001) 34–43
8. Bordogna, G., Pasi, G.: Flexible querying of structured documents. In: Larsem, H.L. et al. (eds.): *Flexible Query Answering Systems, Flexible Query Answering Systems, Recent Advances*, Physica-Verlag, Springer (2000) 350–361
9. Borlund, P.: The IIR evaluation model: a framework for evaluation of interactive information retrieval systems. *Information Research*, 8(3), (2003) paper no. 152
10. Ciravegna, F., Dingli, A., Petrelli, D., Wilks, Y.: User-System Cooperation in Document Annotation based on Information Extraction. In: Gómez-Perez, A., Benjamins, R. (eds.): *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, Lecture Notes in Artificial Intelligence Vol. 2473. Springer-Verlag, Berlin Heidelberg New York (2002) 122–137
11. Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H.P., Studer, R., Witt, A.: On2broker: Lessons Learned from Applying AI to the Web. Research report no. 383, Institue AIFB, Karlsruhe University (1998)
12. Dieter Fensel: Language Standardization for the Semantic Web: The Long Way from OIL to OWL. *Proceedings of the 4th International Workshop on Distributed Communities on the Web*, Lecture Notes in Computer Science Vol. 2468. Springer-Verlag, Berlin Heidelberg New York (2002): 215–227
13. García, E., Sicilia, M.A., Díaz, P., Aedo, I.: An Interactive Ontology-Based Query Formulation Approach for Exploratory Styles of Interaction. In: Jacko, J.A., Stephanidis, C. (eds.): *Human Computer Interaction. Theory and Practice*. Lawrence Erlbaum (2003)
14. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 2003 (to appear).
15. Labrou, Y., Finin, T.: Yahoo! as an Ontology: Using Yahoo! Categories to Describe Documents. In: *Proceedings of the Eighth International Conference on Information Knowledge Management (1999)* 180–187

16. Lynch, P.J., Horton, S.: *Web Style Guide: Basic Design Principles for creating Web Sites* (1999) Yale University Press
17. McCarthy, J.: Approximate Objects and Approximate Theories. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.): *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning* (2000) 519–526
18. Papazoglou, M.P., Porpoer, H.A., Yang, J.: Landscaping the Information Space of Large Multi-Database Networks. *Data & Knowledge Engineering*, **36**(3) (2001) 251–281.
19. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, Riedl J.: GroupLens: An open architecture for collaborative filtering of netnews. In: *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, Chapel Hill, NC: ACM (1994) 175–186
20. Robins, D.: Interactive Information Retrieval: Context and Basic Notions. *Informing Science Journal*, **3**(2) (2000) 57–62
21. Rosenfeld, L., Morville, P.: *Information Architecture for the World Wide Web*. O'Reilly (1998)
22. Sicilia, M.A., García, E., Aedo, I., Díaz, P.: A literature-based approach to annotation and browsing of Web resources. *Information Research Journal* **8**(2) (2003)
23. Spink, A., Saracevic, T.: Human-computer interaction in information retrieval: nature and manifestations of feedback. *Interacting with Computers*, **10**(3) (1998) 249–267
24. Spink, A., Wolfram, D., Jansen, B. J., Saracevic, T.: Searching the Web: The public and their queries. *Journal of the American Society for Information Science and Technology*, **52** (3) (2001) 226–234
25. Stuckenschmidt, H., van Harmelen, F.: Approximating Terminological Queries. In: Andreasen, T., Motro, A., Christiansen, H., Legind-Larsen, H. (eds.): *Flexible Query Answering Systems. Lecture Notes in Computer Science*, Vol. .2522. Springer-Verlag, Berlin Heidelberg New York (2002) 329–343
26. Toms, E.: Serendipitous Information Retrieval. In: *Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland European Research Consortium for Informatics and Mathematics (2000)
27. Voorhees, E. M.: Using WordNet for text retrieval. In: Fellbaum, C. (ed.): *WordNet: An Electronic Lexical Database*, MIT Press (1998) 285–303

Usability Issues Confronting Mobile Devices as Internet Interfaces for General Purpose Navigation

José María Gutiérrez, Roberto Barchino, and José Antonio Gutiérrez de Mesa

CS Department, Alcalá University, Alcalá de Henares, Spain
{josem.gutierrez, roberto.barchino, jantonio.gutierrez}@uah.es

Abstract. Mobile devices are the newest Internet Clients but they are quickly achieving an important position due to the use of great number of these devices around the world. Companies which grant access to Internet, the net of nets, to this new kind of net, are pushing hardly, but they do it in order to get more and more profit. We suspect that in this fast evolution the main subject has been forgotten, that is the user interest. A lot of emphasis has been done in marketing and technology but, does the user get all the things companies promise?. In this paper we analyze the interfaces provided to the users and how them affect the usability of Internet through these devices, from our experience in different related fields, Mobile Devices Programming, Web Interfaces Adaptability or Internet Basics.

1 Introduction

At the present time, we can consider Internet development evolution is in saturation phase [1] (see Fig. 1). Although this technology is still being developed, many people consider it as the base technology for creating higher level technologies. So, now the emphasis is in these higher level technologies.

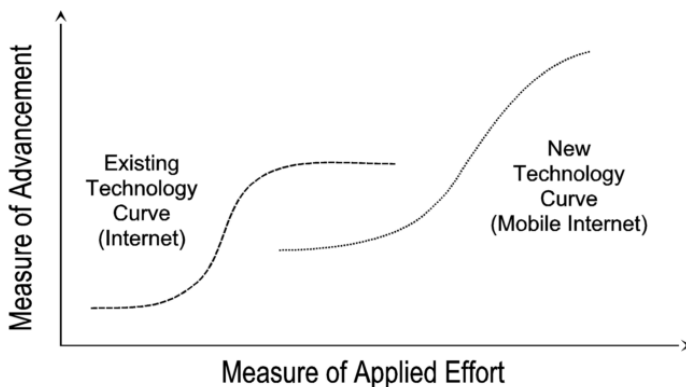


Fig. 1. S-Curve for new al existing technologies, which relate effort and advancement

It is interesting to consider from an historic point of view that there have been technologies which represented an advance for the human civilization, this includes Internet. All of these Technologies with the passage of time became a daily fact, a normal and necessary tool for the social construction and maintenance. Examples of this kind of technologies are water conduction, gas conduction, electricity, and, more recently the telephone and radio and television broadcasting. All of these technologies are assumed today like basics needed for the support of modern society. Internet, depending on the country and continent (unfortunately like the others) is reaching this status of basic technology for society.

In this frame, there are two main tasks to fulfill to keep on developing Internet. The first one is the net development itself. The second one is the advance in the creation of new technological ideas based on the network.

In Science Computer Department in Alcalá University there are several teacher teams with shared members among them which have worked in the last few years in Internet areas. Works about the net and about areas based on it or built directly on itself have been realized [2, 3]. Works about adaptability interfaces, internationalization interfaces and disabled people access to interfaces have been developed.

In SC Department, we have also realized works about the new technologies of third (3G) and 2.5 (2.5G) generation mobiles [4].

The present work seems very disparate in the beginning from those research thread. It shows the confluence of the accomplished works for Internet about classic interfaces with works on these new mobile interfaces and the usability and accessibility problems for users accessing through them.

Since the moment of appearance of 2.5G services, the introduction of WAP service (Wireless Access Protocol) [5] which allows the Internet access from a mobile terminal is being tried. This technology had not the expected success [6, 7] because of its high costs and advertising campaigns which offered the service as more advanced user experience than it was.

Now with the GPRS introduction (other 2.5G service) [8] the costs have been reduced considerably. With GPRS and contents improvement, WAP starts to spread, having the apparition of real access from mobiles to Internet.

Our pursuit, participation and promotion of the work about the line of programming and use of the 2.5G terminals, allow us to have a proper position to discuss from the usability experience and the necessary or possible improvements in these terminals interfaces.

Based in all those previous works, we will study terminals usability and their interfaces for an easy navigation through Internet. We will also analyze the availability of these terminals for handicapped users.

Finally we will propose some advices to increase the comfort and friendly use of the interfaces in order to achieve extensive use of these terminals for Internet access which is the main goal the industry seeks and is still very far away.

2 Actual Mobile Phone Interface

Every machine interface which allows human interaction has two control sorts; input and output (from the machine point of view). The input interface for a mobile device

is its keyboard, and actually the only one. In the near future, navigation with voice or by means of pointed devices will be added although.

It is true that PDA include as part of they interface a tactile screen and a pointer which let us to compare it with the mouse for the navigation in Internet. The output interface is a small graphic screen which in the last versions is able to draw colors. Also in this case, sound communication could be used but there is no still news of its used in terminals or systems.

In this point we must mark that along this work we are mainly referring to mobile telephones. The medium PDA which includes a mobile connection can be consider as hand computer with wireless connection instead of mobile device which offers less computing, memory and interface capabilities.

There are several works which study the screen utility and efficiency of these devices [9] and the existing improvement possibilities. These works are based in the existing limitations due to the small size screens, the limitations in colors and the effect in the device productivity of these characteristics. These limitations will determine that device will be used or not by the users to access to advanced services as WAP.



Fig. 2. Mobile Phone Emulator with Interface Devices Indicated

The interface which allows the user to transfer demands to the device, the keyboard, has been less studied in these works. Probably, due to the fact that the classic works about usability in general web systems have been focused on the graphical interface. This is because the input interface was successfully implemented with many standard devices. On the other hand, in mobiles devices, this interface is far of being enough perfected to allow a handy use (ergonomic) by the users of these services.

2.1 Mobile Phones Input Interface

On these devices, we have the keyboard as the main interface.

If we analyze historically the device interface development, we have the following stages:

- A first stage in which the phones only handed a numerical keyboard and some special keys because it was only possible to send and receive voice calls.
- This keyboard was growing to contain letters in the same buttons than the numbers to let the storage of names in the phone notebook. Some key other keys were added to simplify the navigation in the notebook.
- Then, the small messages service (SMS) came, the keys with letters were then used to create messages. The navigation keys extended their function towards the navigation on a menu system growing more complex. The frequent use of this kind of service showed that the keyboard was not efficient to create the messages. A very positive improvement for the user was introduced then, the text predictive writing, although the users planted their own solution with the creation of all kind of abbreviations for the words, reducing and adapting them to the keyboards shape. This user's option is also useful for them because they can include more words in a message which have a maximum of 160 characters.
- In the present stage, we have access services to Internet from the mobile, i-Mode and WAP. Also multimedia services with multimedia messages adding photos, video and their creation with video cameras which in the terminal and touching-up programs. To use these services, only the keyboard is not good enough as a tool by several reasons. Firstly, the keyboard has not enough sensitivity to manipulate an image pixel by pixel. In fact, the keyboards are thought to face a hostile environment, like the user pocket, a handbag and others, and there are models which can resist liquids immersion. These keyboards have not the sensitivity required and there are no rules to issue the necessities of specific keys to the required applications by these new services. There are also models which have special keys, or covers which slide to hide or to show the keys, but these covered keyboards continue being equal strong or rough. To finish, there are keyboard which have multidirectional keys or joysticks, but we continue with the same problem because of the sensitivity lack (see Figs. 3, 4 and 5). The most adapted models, in this way, incorporate pointers and tactile screen, but their size and weight makes them to be nearer of a PDA but neither with the same capacities than these ones nor their high value. (see Fig. 6).

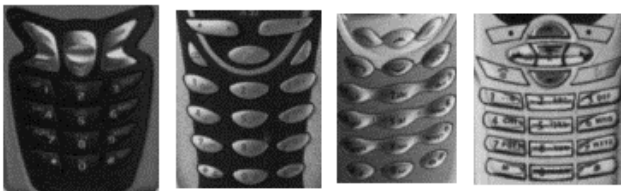


Fig. 3. Mobile phones with “only buttons” interface

The answer to these problems is not new in the interfaces development. In fact, the problems related with these devices, are problems which appeared time before in the computer general evolution. So, the solutions are the same which have showed their utility in general purpose computers, and they are:

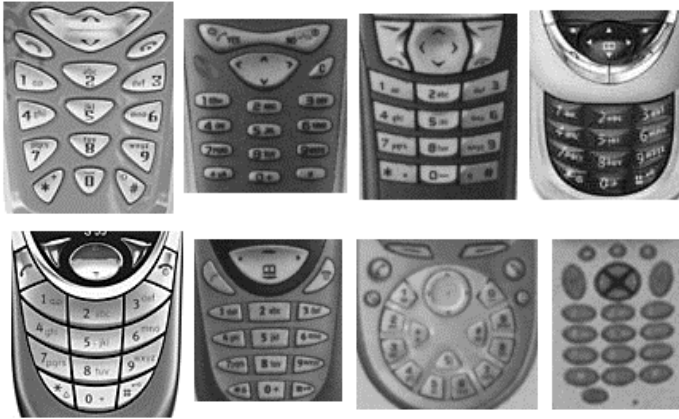


Fig. 4. Mobile phones with “direction button” interface

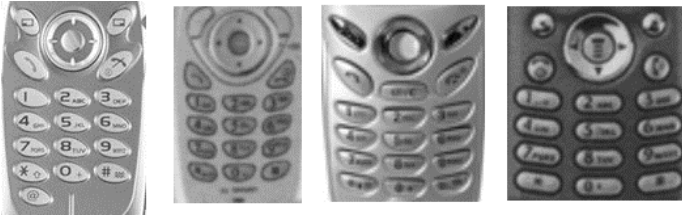


Fig. 5. Mobile phones with “circular joystick button” interface



Fig. 6. Other Mobile Phones with adapted interfaces

- To carry out improvements in the keyboard and in the rest telephone elements to achieve a greater flexibility and comfort in their use for general tasks.
- To create or adapt external devices like navigation mouse in a similar way as it is done with the camera to take photos which are external devices. These external devices can be a mouse or trackball in a small size, easy to carry and connect to the mobile terminal. This kind of devices must have a good industry backing to avoid the multiple kind proliferations of connectors and communication formats which would rend them impossible from an economic point of view.

- To design telephones with PDA functions, smaller and comfortable to carry than the PDA, but more flexible and powerful than the present telephones. There are telephones of this type, but they have the same size that the PDA and are too great with so the telephone advantage is lost.
- Navigation by voice. This option can be very interesting although perhaps a greater calculation power in processors of telephones is needed and to establish common rules which give security to the manufacturers about the effort they are going to realize is useful.

An new devices example with more adaptation to user necessities can be Hand-spring's Treo 600 (Fig. 7), but this device do not comply with the needed specifications for Mobile Internet. This device is a new and small PDA with a fully *qwerty* keyboard.

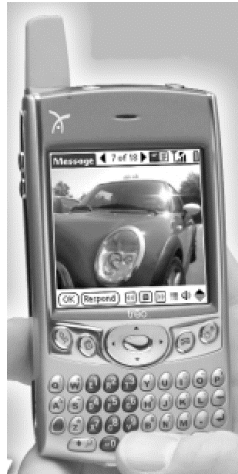


Fig. 7. PDA fashioned like Mobile Phone and including QWERTY keyboard

2.2 Mobile Device Interface and Handicapped People

In all the related works it is made a special notation in the subject of Mobile Phone Internet Access called to be the generalized method to access to the net of nets. This is summed in the sentence “Mobile Internet is Internet for all”.

We have faced interface problems, centered in the input interface. We did it from the perspective of comfort and usability, assuming in every case that the interface allows access and use of the contents of the chunks of information that reside in Internet with the telephones format need.

These problems cause the motto “Internet for all” not fulfilled, since the users are able to use these devices for accessing the net, but the effort involved in its difficult use prevents them from using it, this has been demonstrated in several studies [10].

This stands for general public, but, what happens with the handicapped people? How much influence is a handicap in the use of this interface?

Multiple possible handicaps exist, but from the point of view of the interface of the mobile device we can simplify reducing the study to the handicaps that limit the reception of the visual, sound and tactile information produced by the terminal and the introduction of information in the terminal, made by means of tactile, visual and sound interaction. Therefore we will consider visual and listening handicaps.

In the case of the people with auditory handicap, the main problem that can be found in the use of the mobile phone is the normal communication that is the use of the voice. Navigation by Internet is visual almost in its totality in the interaction of the user and in the access to information. Therefore, the new services do not create problems for these users. Even though it's interesting to raise the possibility of using the services to improve the phone utility. The conflicting service could be replaced of several forms:

- The first solution consists on the use of the answering service by someone without the auditory limitations to create a voice message indicating that the terminal belongs to a user with auditory problems and asked for the shipment of text messages instead of voice calls. This simple system leans in the text messages and the answering machine. The implementation cost is null, but it has the problem of not solving the service completely. In the case of the calls made from telephones that are not mobile, it is still not extended the possibility of shipment and reception of text messages.
- The second proposed solution would be able to totally replace totally the service of voice calls by means of text messages service. This solution would imply the telephony operator which gives service to the handicapped user. In order to obtain it is necessary the operator knows the user who is acquiring a terminal and a telephone number presents a handicap. The operator will have to be in charge to receive all the calls towards that number in a voice mailbox, to process them and to transform them into text messages that will be sent to the user. In addition the user will need a special number from the operator to which to send text messages with a certain format. These messages will be transformed in voice messages from the original text message using a voice synthesizer. This voice message will be sent by means of a voice call to the number indicated as a part of the user text message. This solution presents some technical problems (voice analysis to generate text) and some special development necessities by the operator, who must see a profit to implement them, this is the reason why it is more difficult then to implement.

In the case of the people with visual deficiencies, the basic problem found is that they cannot know the state in which is the telephone, within its different options. This problem could be solved in an easy and simple way if the mobile phone indicates with certain sonorous messages when a key is pressed or it is acceded to an option of the menus. By sound the person must know which key he has pressed.

For example, when a user with a visual deficiency wants to send a SMS through a mobile phone, the solution will be the following, when it is acceded to the menu; the device reproduces several sonorous aid messages to determine in any moment the situation in the menu of options. The pressing keys during the message body writing must generate a sonorous message which would indicate him the pressed letter.

But it won't be enough with simple sonorous information, because when the telephone receives some messages or when the user interrupts temporally the use of the terminal and the situation in which it is within the menus of the device is not known. The device must implement several added functionalities. The telephone must have a voice synthesizer to reproduce the text messages, with special attention to the messages with abbreviations that could cause an inexact or non understood reproduction by the user. This synthesizer will also serve to solve the problem of which the user does not know at a certain moment the state of navigation among the different menus from the terminal. The device must have a key which informs to the blind user with a message where it is established the phone situation and then can continue with the predicted action. This would have to be translated in the incorporation to the mobile device of new keys.

The access to Internet (WAP) can be feasible for the blind users, extending the presented synthesizer capacities. Then sonorous messages of menus WAP and the chosen options by the user will be generated. The solution must consider how to inform about the new situation to the blind user, for example: the existence of a link in the WML page, the WAP server change caused after clicking a link, etc.

These recommendations we raised would enable to the blind users to be able to use and to obtain all the functionalities from the mobile phones.

3 Conclusions

The main conclusion we can obtain is that the purpose "Internet for all" which lies under the intents of extending Internet access from mobile phones (WAP) is still a hope. Interface difficulties, the lack to obtain a really useful keyboard and a big enough screen, make this objective to be very far away for the moment.

We have centered this paper in the problems with the input interface. We have analyzed the problems and reasons why this interface makes, not impossible, but improbably that common users want to use these devices to access Internet. Also, we took a look to the necessities of handicapped people in their use on these devices in general services of navigation services.

So, as second conclusion, we affirm that to achieve this objective, industry needs to stop the present lines of work to dedicate particular efforts to get a correct interface. This is because all companies in this industry dedicate most of their effort to create services which can be new and amazing but mostly useless. And it would be better if they tried to improve the potentially useful existing services improving the accessibility and usability of the interfaces, it would produce profits for users and companies, and it would cover some existing and real necessities.

References

- [1] Michael S. Slocum, *Technology Maturity Using S-curve Descriptor*. The Inventioneering Company. Costa Mesa, CA.

- [2] Barchino, R. et al., 2001. EDVI: Un sistema de apoyo a la enseñanza presencial basado en Internet. *VII Jornadas de Enseñanza Universitaria de la Informática*. Mallorca, Spain, pp. 451–453.
- [3] Barchino, R. et al., 2003. *Scoring algorithms for Evaluation tests in advanced e-learning systems*. IADIS International Conference e-Society 2003 . Lisboa, Portugal.
- [4] Mobigame, 2003. <http://www.mobigame.uah.es>
- [5] WAP: Wap Forum, 2003. <http://www.wapforum.org>
- [6] Nielsen, J. (1999). “Graceful degradation of scalable internet services, WAP: wrong approach to portability,” Alertbox 31/10/1999 at www.useit.com/alertbox/991031.html
- [7] Nielsen, J. (2000). “WAP backlash,” Alertbox 09/07/2000 at www.useit.com/alertbox/000907
- [8] GPRS, 2003. <http://www.mobilegprs.com>
- [9] Dillon, A., Richardson, J. & McKnight, C. (1990) “The Effect of Display Size and Text Splitting on Reading Lengthy Text from the Screen,” *Behavior and Information Technology*, 9(3):215–227 Annual Meeting, 1:356–359
- [10] Nielsen Norman Group Report. Authors: Marc Ramsay and Dr. Jakob Nielsen. WAP Usability Report December 2000

Navigating through the RLATES Interface: A Web-Based Adaptive and Intelligent Educational System

Ana Iglesias, Paloma Martínez, and Fernando Fernández

Computer Science Department
University Carlos III of Madrid
Avda. de la Universidad, 30, 28911-Leganés (Madrid), SPAIN
Tel: 34-91-624{9421, 9454, 9105}, Fax: 34-91-6249430.
{aiglesia, pmf, ffernand}@inf.uc3m.es

Abstract. The paper shows the architecture of the RLATES system, an Adaptive and Intelligent Educational System that uses the Reinforcement Learning model (RL) in order to learn to teach each student individually, being adapted to their learning needs in each moment of the interaction. This paper is focused on the interface module of RLATES, describing how the student could navigate through the system interface and how this interface adjusts the page contents according to the user learning needs. For this adaptation, the system changes the links appearance of the page and the presentation of the system knowledge.

1 Introduction

Web-based education (WBE) is currently a hot research and development area. They have two useful benefits: classroom independence and platform independence. Traditional web-based courses usually are static hypertext pages without student adaptability, providing the same page content and the same set of links to all users. However, since last ninetieths, several research teams have been implementing different kinds of adaptive and intelligent systems for WBE [1].

Web-based Adaptive and Intelligent Educational Systems provide intelligence and student adaptability, inheriting properties from *Intelligent Tutoring Systems (ITS)* and *Adaptive Hypermedia Systems (AHS)*. These systems are the most complete up to now, because the domain knowledge, the student model and the pedagogical strategies used to individually adapt to the student are studied by Intelligent Tutoring Systems, and the application of different forms of student models in order to adapt the hypermedia pages content and links showed to the students (adaptive presentation and adaptive navigation support) are studied by Adaptive Hypermedia Systems. On the one hand, "Intelligent Tutoring Systems (ITSs) are computer-aided instructional systems with models of instructional content that specify *what* to teach, and teaching strategies that specify *how* to teach" [17]; and on the other hand, Adaptive Hypermedia Systems

adapt the content of a hypermedia page to the user's goals, knowledge, preferences and other user's information for each individual user interacting with the system [1].

The first Web-based AIES appeared in 1995 and they arose from Intelligent Tutoring Systems. Some examples of this systems are ELM ART [3], InterBook [2], CALAT [12] or WITS [13]. Subsequently new intelligent and adaptive techniques have been introduced in these systems, as the adaptability of the pages content adjusting them to the user needs in each moment of the interaction (Adaptation Hipermedia) [6][13] and collaborative adaptation in web [11], among others.

The RLATES system (*Reinforcement Learning in Adaptive and intelligent Educational System*) is a Spanish DataBase Educational System in Web that uses the Reinforcement Learning model in order to provide the student an "optimal" curriculum sequence according to his learning characteristics in each moment of the interaction. At the same time, the system uses adaptive techniques in order to adjust the presentation of the content to the student and to help him to navigate through the application interface. The RLATES system forms part of the PANDORA project¹ [5], whose main goal is to define methods and techniques for database development implemented in a CASE tool, useful for students and practitioners.

The paper is organized as follow: first, the proposed architecture of the AIES is briefly defined in section 2. In section 3, the navigation through the RLATES interface is described and finally, the main conclusions and further research of this work are given.

2 System Architecture

A typical structure of an ITS, and hence, of an AIES, is composed of four well differentiated modules [4] (see Figure 1).

The *student module* contains all important information about the student in the learning process: goals, student background knowledge, personal characteristics, historical behavior, etc. It is necessary, for the effectiveness of RLATES, to construct a good *student model* and to classify learners according to their critical characteristics in learning the system knowledge. This classification is very important for the pedagogical model, as we will see in following paragraphs.

The *domain module* contains all characteristics of the knowledge to teach. The traditional knowledge structure (hierarchical structure) could be an advantage in the pedagogical strategy in AIES, where each topic has been divided into sub-topics, and these in others sub-topics, and so on. This structure implies "is-parent" relationships between the topics. An other relationship could be applied to the knowledge, the "pre-requisite" relationship, where the *A* topic is pre-requisite of the *B* topic if the tutor can not show the *B* topic when the *A* topic has not yet been shown. At the same time, each node of the tree contains tasks (sets of definitions, examples, problems, exercises, etc.) in several formats (image, text, video, etc.).

¹ CASE Platform for Database development and learning via Internet. Spanish CICYT project (TIC99-0215).

The *pedagogical module* decides *what, how* and *when* to teach the domain module contents, taking the better pedagogical decisions according to the user needs. The definition of this problem as a Reinforcement Learning problem is described in previous papers [9][10], where the system perceives the current student's knowledge by evaluations (*tests*) and choose the next action to be executed according to the current *action policy, B*. The *action* corresponds with showing set of leaves of the knowledge tree (*definition, exercise, problem, etc.*). When the action is executed, a state transitions it is supposed to occur, and the environment provide a *reinforcement signal* (positive or negative) to the system. The reward signal supplies a maximum value upon arriving to the goal of the tutor. This signal is used to update the system's action policy. The system behavior, *B*, should choose the actions that tend to maximize the long-run sum of values of the reinforcement signal, choosing in this way the optimal tutoring strategy (what, when, and how to teach; the best sequence of contents and how to teach them) to coach the current learner. The value-action function $Q(s,a)$

estimates the usefulness of executing one action (showing leaves of the knowledge tree to a student) when the system is in certain knowledge state. Therefore, the goal of the learning process is to find the policy that maximizes this function.

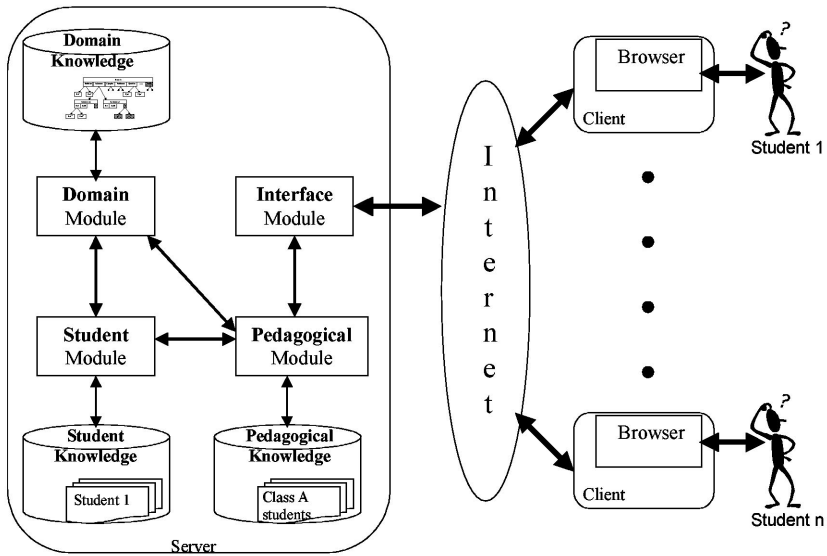


Fig. 1. RLATES Architecture

Updating the action policy can be performed by different algorithms. The algorithm implemented in RLATES is the Q-learning algorithm, where its value-action function is defined in the equation 1.

$$Q(s,a)=(1-\alpha) Q(s,a)+ \alpha\{r+ \gamma \max_{a'} Q(s',a')\} \quad (1)$$

This equation requires the definition of the possible states, s , the actions that the agent can perform in the environment, a , and the rewards that it receives at any moment for the states it arrives to after applying each action, r . The γ parameter controls the relative importance of future actions rewards with respect to new ones, and α parameter is the learning rate, that indicates how quickly the system learns. In [9] how the Q-learning algorithm is adapted to the Adaptive and Intelligent Educational System domain is explained.

Finally, the *interface module* facilitates the communication between the AIES and the student. This module applies intelligent and adaptive techniques in order to adapt the content and the navigation to the students, leaning on the pedagogical module, that decides which is the next task to be showed to the student and in which format the knowledge is going to be taught.

This article focuses on the description of the interface module, since previous articles [10] have shown that the application of the reinforcement learning model in the pedagogical module permits that the system learns to adapt its pedagogical strategies according to the student needs in each moment, only based on information of the interaction with other students with similar learning characteristics. Besides, the theoretical model have been successfully applied to de DataBase Design domain [4], showing subsequently that the AIES can learn an optimal policy to teach students interacting with reasonably few students [5].

The intelligent and adaptive techniques used at the interface model of the RLATES system are described in next section, where we have distinguished between the adaptive navigation support and the adaptive presentation of the system contents.

3 Hypermedia Adaptation in RLATES

The Adaptive Hypermedia Systems (AHS) provide a not lineal progression through the material of the AIES, where a *hyperdocument* consists of a set of pages connected by links. In previous hypermedia systems this property implied a problem: the student used to loose himself in the hyperspace, with no idea of what to study next and how to return to the previous page learned [15]. The new hypermedia systems resolve this problem adding an intelligent component to be adapted to the users and to help them. On the one hand, these systems could adapt the navigation of the students (*Adaptive Navigation Support*) helping them to find their paths in hyperspace by adapting link presentation and functionality of the pages according to the goals, knowledge and learning characteristics of an individual user.

On the other hand, the system could adapt the content of a hypermedia page (*Adaptive Presentation*) according to the goals, knowledge and other characteristics of the student.

In this section how the RLATES system adapts to user is described, analyzing the two major technologies in adaptive hypermedia: adaptive navigation support and adaptive presentation.

3.1 Adaptive Navigation Support

“The goal of the *Adaptive Navigation Support (ANS)* technology is to support the student in hyperspace orientation and navigation by changing the appearance of visible links” [1]. This technique shares the same goal that the technique of *Curriculum Sequencing* in Intelligent Tutoring Systems: to provide the student a sequence of knowledge items (knowledge units) and tasks (examples, questions, problems, etc.) to learn adapting this sequence to the learning characteristics of each individual student. That is to say, the system helps students to find an “optimal path” through the AIES learning material. But, at the same time, the ANS technique adds some advantages of the hypermedia context: it can guide the students directly (like curriculum sequencing), but also indirectly (by the presentation of the links in the page).

Next, the ways the RLATES system adapts the links are described, illustrated by several figures:

On the one hand, the system can **guide** the student **indirectly** through the AIES content by changing the appearance of the links in the page, changing the color or the order of the links in the page. The RLATES system includes a table of the AIES contents (in a tree format) always visible in the left frame of the application that provides the indirect guide to the users. This tree of contents has one link for each one unit of knowledge in the domain module of the AIES. If the student click one of them, the system will show at the right frame tasks of the item of knowledge. In Figure 2 we can observe how the system shows some tasks of the “Entidades” (*Entities*) item, chosen by the student.

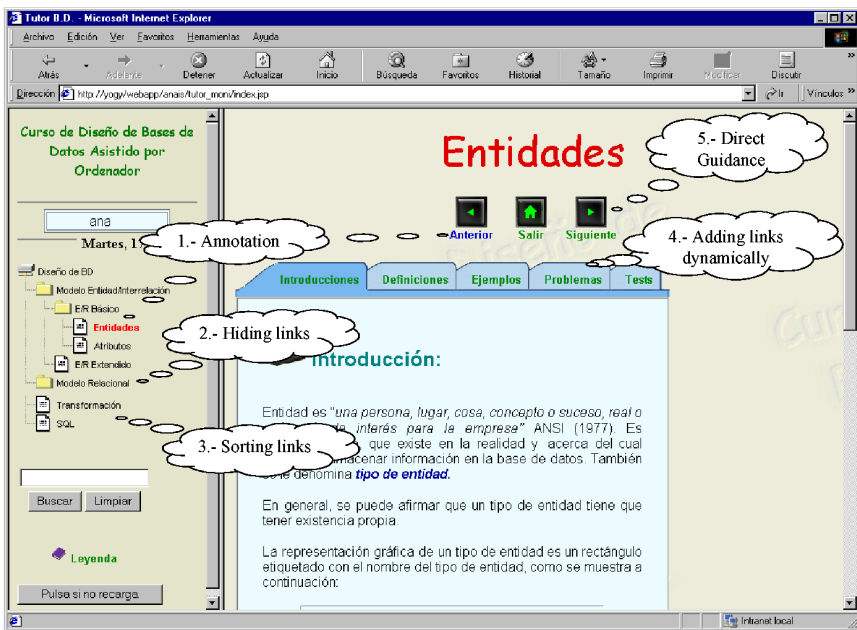


Fig. 2. Techniques of link adaptation in RLATES system

The RLATES system uses the following indirect techniques of guidance that we can see in an example in Figure 2:

1. **Annotation:** Is the most popular form of ANS. With this technique, the system augments the links with some comments about the pages behind the annotated links. The RLATES system changes the color of the links of the tree of contents according to the student historical behavior (see the Figure 3); the name of the link is bolded and the color is red if the topic is the actual topic (we can show tasks of this topic in the right frame); the color of the topic is black if the topic have not been shown yet; the color is green if the student has passed the last test about this topic; and finally, the color is orange if the student has not passed the last test about this topic.
2. **Hiding links.** This technique hides the topics that at a certain moment are not relevant for the student. For instance, when the student is studying the “Entidades” (*Entities*) topic of the “Modelo Entidad/Interrelación” (*Entity/Relationship Model*) topic, the units of knowledge of the “Modelo Relacional” (*Relational Model*) topic are not relevant for the student at this moment.
3. **Sorting links.** This technique tries to physically order in the page the links according to the model of the user (normally based on the student knowledge or in the student behavior). The RLATES system order the topics in the knowledge tree according to the “pre-requisites” and “is-parent” relationships of the units of knowledge described in the AIES domain module.
4. **Adding/Removing links dynamically.** This technique adds or removes dynamically links to the page according to the learning needs of the student. It tries to prevent the user from following links that are not relevant for him/her at a certain moment. RLATES adds or removes links of the right frame of the application according to the tasks stored at the domain model about this particular topic. For instance, if the “Entidades” (*Entities*) topic has not definition tasks, the link “Definiciones” (*Definitions*) will be removed from the page.

On the other hand, the system can guide the student through the material by **Direct Guidance** when the student chose the “next” button to continue the learning process (see the Figure 2). When this situation happens, the system drives the student to the “best” page in the hyperspace according to their learning characteristics (current knowledge, goal, etc.). The task of choose the next page to be shown (the next action to execute) concerns to the pedagogical module of the AIES. This module uses the reinforcement learning model in order to adopt a decision. A great variety of exploration/exploitation strategies could be used in reinforcement learning problems in order to choose the next action to execute [184]. For example, the *e-greedy* strategy tries to select the action with greater value of the Q function when the system is at the s state ($Q(s,a)$) with a probability of $(1-e)$. That is to say, the e parameter shows how greedy is the system when it chooses a new action to execute: when $e=1$, the system randomly selects the next action to be executed and when $e=0$ the system chooses the action with a greater value of Q . Another very common exploration/exploitation strategy is the Boltzmann exploration policy, that estimates de probability of choosing the

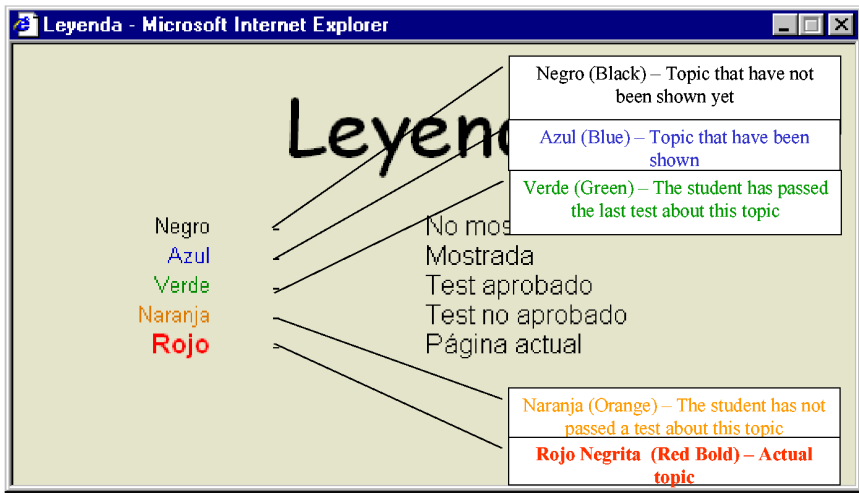


Fig. 3. Link annotation in RLATES system

action a according to the next function, where τ is a positive parameter called the *temperature* and $Q_t(x)$ is the value of the Q function at the t time when the system is in a certain state s and the action x is executed:

$$P(a) = \frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b=1}^n e^{\frac{Q_t(b)}{\tau}}} \quad (2)$$

If the temperature is high, all the probability of the actions have almost similar values and if the temperature is low, it causes a great difference in selection probability for actions.

In distance educational system (like our system), to maintain the attention of the user is absolutely necessary, and to give the sensation that the student has the control of the interaction is very important too. This is why RLATES has implemented the *direct guidance* by proposing the student more than one possibility for the next page (at almost 10 possibilities). Then, the student is in charge of choosing the next page to visit, bearing in mind the suggestion of RLATES that appears at the page in percentage format. In Figure 4 we can see the page that appears when the student clicks the “next” button in our application.

The RLATES system implements the Boltzmann exploration/exploitation policy, because it provides the probability of choose an action to be executed, not as the *e-greedy* exploration/exploitation policy. In [7], some experiments analyzing the differences between the Boltzmann and the *e-greedy* exploration/exploitation policies applied to Adaptive and Intelligent Educational Systems have been done.

3.2 Adaptive Presentation

“The goal of the *Adaptive Presentation* technology is to adapt the content of a hypermedia page to the user’s goals, knowledge and other information stored in the user model” [1]. RLATES implements this adaptation too by using the reinforcement learning model. The tasks of the topics are stored in the domain model, keeping information about the formats of their pages. When the student clicks the “next” button, the system will choose a task in a format that adjusts to the student learning needs.

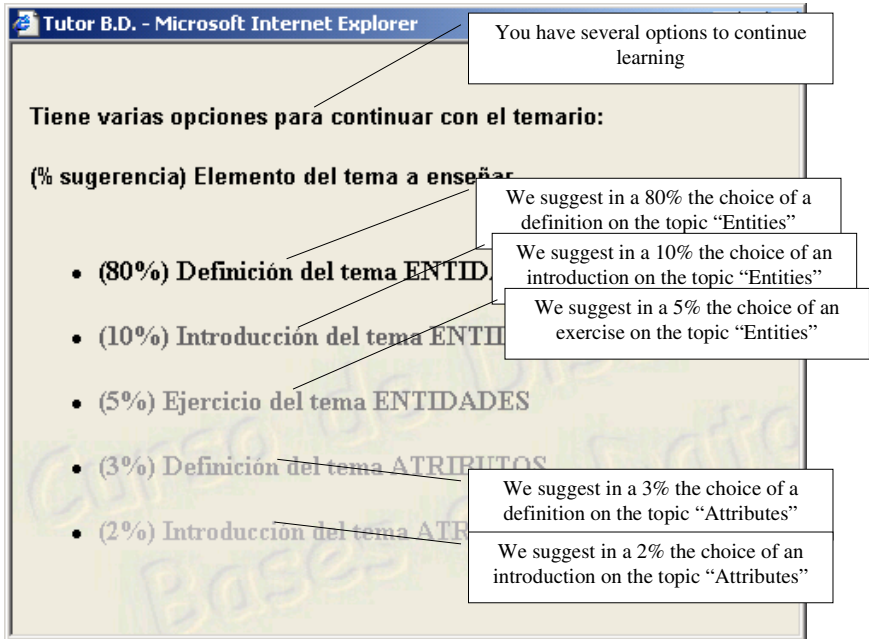


Fig. 4. Hypermedia page where the student can choose the next task of RLATES to be executed.

4 Concluding Remarks

This paper describes the architecture of the RLATES system, focusing on the interface module. How the RLATES system uses intelligent and adaptive techniques in order to adjust the teaching according to the current student in each moment of the interaction has been explained. The system adaptation has been divided in two problems: the *Adaptive Navigation Support*, where RLATES guides the user both, by *direct guidance* (by clicking the “next” button”) and by *indirect guidance*, changing the appearance of the links in the page (by annotation, sorting links, hiding links, adding and removing links, etc.) and the *Adaptive Presentation*, where the system is leaned on the pedagogical model of the system.

This system is being implemented as part of a PhD research. Some experiments with simulated students have been done in order to evaluate the convergence of the Q-learning algorithm in Adaptive and Intelligent Tutoring Systems [7][9]. Now we are doing some experiments in order to evaluate the system with real students.

Acknowledgements. We thanks Ana Isabel Sebastián and Mónica Merino for helpings as at the implementation of the web pages of the RLATES system.

References

1. Brusilovsky, P. *Adaptive and Intelligent Technologies for Web-based Education*. Kunstliche Intelligenz. Special Issue on Intelligent Tutoring Systems and Teleteaching. Vol. 4, 1999.
2. Brusilovsky, P., Eklund, J., and Shwarz, E: *Web-Based Education of all: A tool for developing adaptive courseware*. Computer Networks and ISDN Systems. 30, 1–7, pp. 291–300, 1998.
3. Brusilovsky, Peter, Schwarz, Elmar and Weber, Gerhard. *ELM-ART: An Intelligent tutoring system on World Wide Web*. Intelligent Tutoring Systems. Springer Verlag. Ed. Claude Frasson and Gilles Gauthier and Alan Lesgold, pp. 261–269 (1086), 1996.
4. Burns, Hugh & Capps, Charles. *Foundations of Intelligent Tutoring Systems: An Introduction*. Foundations of Intelligent Tutoring Systems. Hillsdale, N.J: Lawrence Erlbaum Associates. 1–19, 1988.
5. Castro, E., Cuadra, D., Martínez, P., and Iglesias, A. Integrating Intelligent Methodological and Tutoring assistance in a CASE platform: the PANDORA experience. In Proceedings of the Informing Science & IT Education Conference. Cork, Irland, 2002.
6. De Bra, P. and Calvi, L. *An open Adaptive Hypermedia Architecture*. The New Review of Hypermedia and Multimedia, pp. 115–139 (4), 1998.
7. Iglesias, A., Martínez, P., Aler, R. And Fernández, F. *Analising the Advantages of Using Exploration and Exploitation Strategies in an Adaptive and Intelligent Educational System*. 2nd International Conference on Multimedia and Information and Comunication Technologies in Education (m-ICTE03). Badajoz (Spain). 2003
8. Iglesias, A., Martínez, P., Cuadra, D., Castro, E. and Fernández, F. *Learning to Teach Database Design by Trial and Error*. 4th International Conference on Enterprise Information Systems. Ciudad Real (España). Pp. 500–505, 2002.
9. Iglesias, Ana, Martínez, Paloma and Fernández, Fernando. *An experience applying Reinforcement Learning in a Web-based Adaptive and Intelligent Educational System*. Informatics in Education International Journal. To appear.
10. Iglesias, Ana, Martínez, Paloma and Fernández, Fernando. *Applying Reinforcement Learning in Intelligent Tutoring Systems*. 4th International Conference on New Educational Environments. Lugano (Switzerland). Pp. 11–14, 2002.
11. Leigh, J., Johnson, A.E. and DeFanti, T.A. Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments. Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM). Conference on High Performance Networking and Computing, San Jose, CA, pp. 1–14, 1997.

12. Nakabayashi, K., Maruyama, M., Koike, Y., Touhei H. and Fukuhara, Y. *Architecture of an Intelligent Tutoring System on the WWW*. Proceedings of the 8th World Conference of the AIED Society, pp. 18–22, 1997.
13. Okazaki, Y., Watanabe, K., and Kondo, H.: *An Implementation of an Intelligent Tutoring System (ITS) on the World Wide Web (WWW)*. Educational Technology Research 19. 1, pp. 35–44, 1996.
14. Sison, R. *Framework for Intelligent Internet-Based Learning Support*. Proceedings of the Ninth International Conference on Computers in Education. pp. 926–931, 2001
15. Stern, Mia Keryn. *Using Adaptive Hypermedia and Machine Learning to Create Intelligent Web-based Courses*. PhD. Directed by: Professor Beverly Park Woolf. University of Massachusetts, Amherst. 2001.
16. Trhun, Sebastian B.. *The role of exploration in learning control*. In David A. White and Donald A. Sofge (Ed.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, New York, 1992.
17. Wenger, E. *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann, 1987.

Searching with Semantics: An Interactive Visualization Technique for Exploring an Annotated Image Collection

Paul Janecek and Pearl Pu

Human Computer Interaction Group,
Swiss Federal Institute of Technology, Lausanne (EPFL)
CH-1015 Ecublens, Switzerland
{paul.janecek, pearl.pu}@epfl.ch
<http://hci.epfl.ch/>

Abstract. Effective opportunistic search in visual interfaces for large information collections, such as documents, web pages, and annotated images, often requires domain knowledge that the information seeker does not have. Relevant semantic information, such as metadata and especially ontologies, are valuable resources for helping the user learn this domain knowledge. In this paper we describe an interactive “focus + context” visualization technique for integrating this semantic information into the representation of the information collection to support effective opportunistic search. We have implemented a prototype using these techniques that integrates WordNet, a general lexical ontology, and a large collection of professionally annotated images. As the user explores the collection, the prototype dynamically changes the visual emphasis and detail of the images and keywords to reflect the relevant semantic relationships. This interaction allows the user to rapidly learn and use the domain knowledge required for effective opportunistic search.

1 Introduction

Exploring a large, unfamiliar collection of information using queries is like ordering at an exotic restaurant without the benefit of a menu. It is very difficult to access or discover the rich selection just outside our limited vocabulary and knowledge, or to develop a coherent model of the global structure. Visual information retrieval interfaces (VIRIs) aid in the exploration task by representing the collection and its structure, as well as maintaining the user’s history of queries and results.

We are investigating how to extend VIRIs to more directly support the variety of strategies that people use for opportunistic search. Many of these strategies require access to domain knowledge that is not modeled in the interface. Our approach is to integrate semantic information, such as ontologies, into the interface to enable them to explore the information collection at a semantic level. In this paper we describe opportunistic search in greater detail, and discuss the

design of visual interfaces for supporting opportunistic search, especially “focus + context” techniques. We then give an overview of a prototype we have developed for exploring a large annotated image collection using the semantic concepts modeled in WordNet.

2 Opportunistic Search

Unlike the single-shot query where a person knows exactly what they are looking for, opportunistic search is more like research. The “information seeker” must learn a set of new concepts and relationships so they can efficiently navigate in an unfamiliar conceptual space, and evaluate the relevance of the information they find. Bates described this process as “berry-picking” because the query is satisfied by the information gathered along the search process (such as references, influential authors, and well-respected resources), rather than a single set of results (Fig. 1 shows an example)[1].

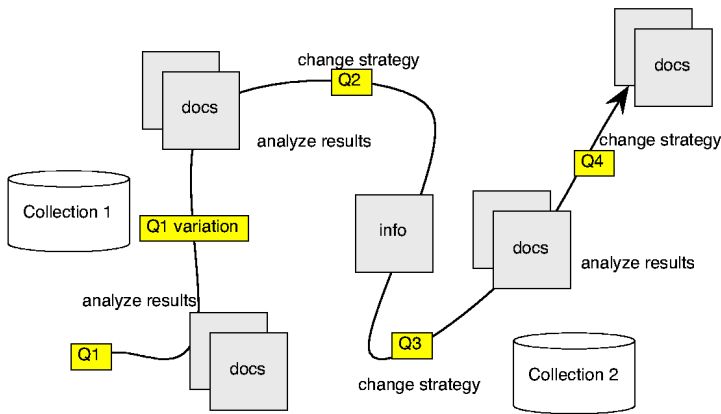


Fig. 1. Opportunistic search is a process of queries and exploration that is dependent on both the information encountered as well as the search strategies used

Bates [2] identified four general types of search tactics that “experts” use to find information:

- *monitoring tactics*, which manage the overall search process (e.g., record promising search paths that have not yet been explored);
- *file structure tactics*, which are techniques for effectively navigating through different information structures (e.g., reviewing all options before committing);
- *search formulation tactics*, which guide the design of the query (e.g., expand a query by adding synonyms); and

- *term tactics*, which guide the choice of words used in a query (e.g., refine a query by selecting a more general or more specific term).

Bates remarks that most interfaces for information retrieval do not directly support these higher-level strategies [3]. Recent advances in VIRIs lay the foundations for enabling these tactics. For example, Furnas’ desiderata for the design of VIRIs [4] suggests visual reification of the “conceptual” aspects of search, such as history, and that information should always make its internal structure accessible for exploration. A VIRI enabled with these properties would facilitate the *monitoring* and *file structure tactics* mentioned above.

However, a significant obstacle that remains is that many of these tactics require access to domain knowledge that is not available in the interface. For example, to apply *search formulation* and *term tactics* the user must know the existence of more general, more specific, or related terms (i.e., the type of knowledge increasingly accessible in metadata and ontologies). Our approach to solving this problem is to integrate this semantic information directly into both the visual and interactive aspects of the interface to facilitate these strategies for opportunistic search.

3 Semantic Visualization and Interaction Techniques

A general problem in visual representations of large collections of information is managing the amount of information that is visible at any given time. Fish-eye view visualization techniques, also known as *focus + context* techniques, are a well-known approach to solving this problem by maintaining a balance between local detail and global structure. We have identified four important general properties that a VIRI should have to support opportunistic search using these techniques [5]:

- *Focus + context*: The balance between the local detail around the user’s focus and it’s context in the global structure must allow the user to perceive potential directions for future search. The focus may be a wide variety of objects in the interface: a query, a set of results, an object or one of its attributes, or perhaps an entire collection of objects. The context is the relationship between the focus and the rest of the information in the collection.
- *Dynamic selection of focus + context*: The user must be able to easily shift both their focus and context to explore different search strategies, (for example, across different domains, levels of granularity, or collections).
- *Multiple foci and multiple contexts*: Opportunistic search is multi-directional. The user must be able to compare the results of different foci and contexts to detect patterns.
- *Flexible modeling of semantics*: As a user explores an information collection, they will gather “interesting” information with undefined or incomplete semantics. The user must be able to externalize their understanding of the information they encounter during the search process in a “light-weight” way.

Our prototype, described in Sect. 4, is being developed to examine the interaction of these properties in opportunistic search.

3.1 Semantic Fisheye Views

Furnas originally defined fisheye views with the following equation [6]:

$$DOI(x|fp = y) = API(x) - dist(x, y). \quad (1)$$

The Degree of Interest (*DOI*) of each object, x , given a focus, fp , is the difference between the *a priori importance* (*API*) of x , and the distance between x and the focus. The visual representation of each object is modified interactively to reflect *DOI*. For example, Furnas implemented a fisheye view using the hierarchy of a structured programming language where *API* was the distance from the root, and *dist* was the path distance between nodes. As the user navigated through the code, lines of code with a *DOI* below a certain threshold were filtered resulting in a very compact display that balanced local detail and global context. However, equation 1 is general and has been redefined in many different data structures such as tables and graphs [7], [8].

In *graphical fisheye views*, the *API* and *dist* are defined spatially within the view or structurally within the model that is being visualized. In the case of opportunistic search, where there are potentially many different types of interrelated data collections, metadata and ontologies, there are also potentially many different types of distance metrics. Therefore, we have generalized equation 1 to include the possibility of balancing multiple weighted distance metrics as shown in equation 2 [9]. This equation adds a weight, w , to each distance function to allow a user to simultaneously explore multiple notions of relevance. *Semantic fisheye views* are based on this framework, with one or more distances defined semantically (e.g., within a related semantic model, such as an ontology), rather than spatially or structurally.

$$DOI_{context}(x|fp = y) = f(API(x), w_i, dist_i(x, y)) . \quad (2)$$

Once *DOI* is calculated, the visual representation is modified using one or more *emphasis algorithms* to allow the user to perceive the relative *DOI* of objects. For example, the size, color, grayscale, or level of detail may be modified, or information below a certain threshold may be filtered. In the next section we describe our prototype based on these techniques.

4 A Prototype for Exploring an Annotated Image Collection

We have implemented a VIRI prototype using *semantic fisheye views* to support opportunistic search in a very large collection of annotated images. Similar to NaviQue [4], the interface allows the user to search for images by defining a query or by browsing the images and their related keywords in a 2D space. The

environment shows a history of previous searches and allows the user to gather interesting images and keywords.

Unlike NaviQue and other VIRIs, however, the prototype uses a related semantic model to support opportunistic search in two different ways. First, as the user explores the collection the *semantic fisheye view* visually emphasizes keywords and images that are semantically related to the current focus. This serves as an explanation of the vocabulary of the collection, as well as a suggestion of potential directions for search that are more general, more specific, or similar to the current focus. Second, the semantic model aids in query disambiguation and, more importantly, query expansion to include related concepts that will return images in the semantic neighborhood of the original query.

In the following sections we introduce the functionality of the prototype with a scenario, describe how the prototype uses a semantic model and interactive visualization to support the *search formulation and term tactics* identified earlier, and a brief discussion of the architecture.

4.1 Searching with Semantics: A Scenario

A mother and daughter are looking for images of horses to use in a scrapbook they are making. They begin the search by typing in "horses", and over 500 images fill the space as shown on the top left of Fig. 2. Although it is easy to zoom in on any group of images, there are too many images and they decide to search for a specific kind of horse. Typing in "Arabian horses" they find the two images shown in the upper right figure. Placing the mouse over one of the images shows different words associated with the image, and placing the mouse over one of the words shows the related images as well as related concepts. When they place the mouse over "Arabian horses", as shown in the bottom left figure, they see the related concepts "Knights", "Bay", "Wild horses" and "Pintos". The daughter wonders what a Pinto is, and placing the mouse over the word she immediately sees images of Pinto horses and decides to include one in her scrapbook. Continuing in this way, they can find other types of horses or concepts they are unfamiliar with or hadn't considered.

In this scenario, the prototype supports opportunistic search by transparently suggesting new directions for their search. Like most VIRIs, the layout places related objects next to each other. The *semantic fisheye view* builds on this by emphasizing objects that are semantically related but not necessarily visually adjacent. The semantic relationships used in the scenario are the term tactics Bates identified [2]:

- SUPER: To move upward hierarchically to a broader term
- SUB: To move downward hierarchically to a more specific term
- RELATE (SIBLING): To move sideways hierarchically to a coordinate term

In the context of the semantic model we use in our prototype, WordNet, these tactics are transitions from a concept to neighboring concepts, as shown in Fig. 3.

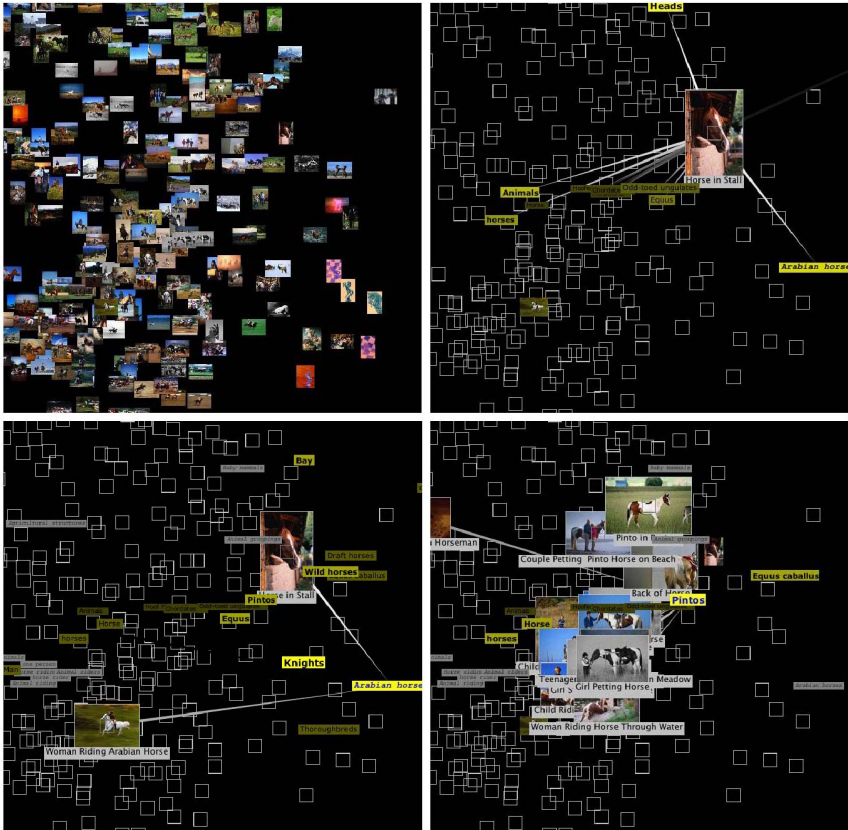


Fig. 2. An example session of browsing the semantic relationships between images

We use a variety of visual emphasis techniques for displaying *DOI* dependant on the type of focus and the distance metrics used to define the context. Table 1 shows several examples of focus, context and emphasis techniques for Images (*I*), Keywords (*K*), and Concepts (*C*).

Table 1. Focus, Context, and Emphasis Techniques

Focus	Context	Emphasis
I	$I \rightarrow K^*$: term order	font size, background shading, edge grayscale
K	$K \rightarrow I^*$: cosine similarity measure	image size, edge grayscale
C	$C \rightarrow K^*$: semantic path distance	font size, background shading

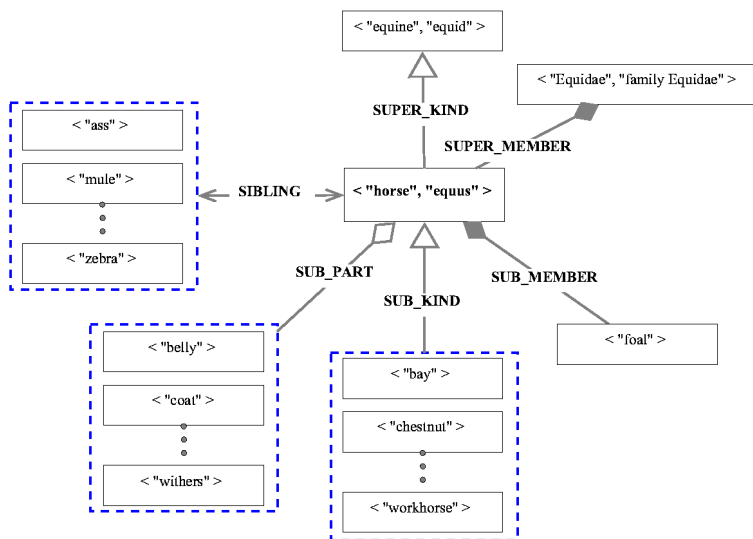


Fig. 3. The SUPER, SUB, and SIBLING tactics using WordNet

4.2 The Corbis Image Collection

On the left of Fig. 4 is a model of our annotated image collection, a set of 56,000 images from the Corbis collection. The figure shows the caption and keywords associated with an image of an Arabian horse. More formally, an image in the Corbis collection, I_i , is represented by an ordered set of one or more keywords, K_i^* , as shown in equation 3.

$$I_i \rightarrow K_i^* = \{k_1, \dots, k_n\} . \quad (3)$$

The images are professionally annotated with an average of over 20 keywords per image describing the objects, actions, and thematic elements of the images. Many of the annotations are hierarchical, meaning that an image of “Horses” will often also include the keyword “Animals” to distinguish it from horses that are not animals (such as “pommel horses”).

4.3 WordNet

WordNet is a large general ontology of the English language developed at Princeton [10]. As shown on the right of Fig. 4, each concept in WordNet is represented by a brief description and a *synset*, a set of synonymous words or lemmas. More formally, a concept in WordNet, C_j , is represented by a set of one or more synonymous lemmas, L_j^* , as shown in equation 4.

$$C_j \rightarrow L_j^* = \{l_1, \dots, l_n\} . \quad (4)$$

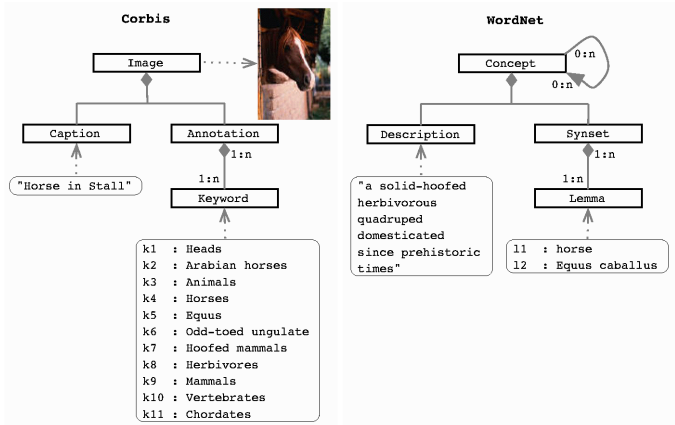


Fig. 4. Models of the Corbis annotated image collection and the WordNet lexical ontology. The concepts in WordNet are organized into a network structure based on a limited set of relationship types

The figure shows the description and lemmas associated with the “horse” type of animal. WordNet is large: there are approximately 140,000 lemmas linked to 110,000 concepts (the same word may have multiple meanings). The concepts are organized into a network using a limited set of semantic and lexical relationships, such as the *Is-A* (hyponym/ hypernym) and *Part-of* (meronym/ holonym) hierarchies for nouns.

4.4 Developing the Correspondence between Corbis and WordNet

In order to explore the image collection at a semantic level (e.g., browse the conceptual relationships between images), we created a mapping between Corbis and WordNet. In our current implementation, the correspondence is only a lexical mapping between the keywords in Corbis (K) and the lemmas in WordNet (L). As shown in Fig. 5, we were able to achieve a correspondence between K and L of over 90% for the 25% most frequent keywords using basic stemming rules (e.g., removing plurals and hyphenated terms). This correspondence drops to under 60% for the keywords that are only used once in the image collection. The majority of these matches are *exact*, for example the keywords “Horses” and “Horse” are both matched to the lemma “horse”. *Inexact* matches are those compound keywords where each subcomponent is exactly matched. For example, the keyword “four-wheeled carriage” is inexactly matched to the lemmas “four-wheeled” and “carriage”.

4.5 Using WordNet to Expand Queries

The imprecise correspondence between Corbis and WordNet introduces multiple levels of ambiguity when trying to search for images using concepts. In practice,

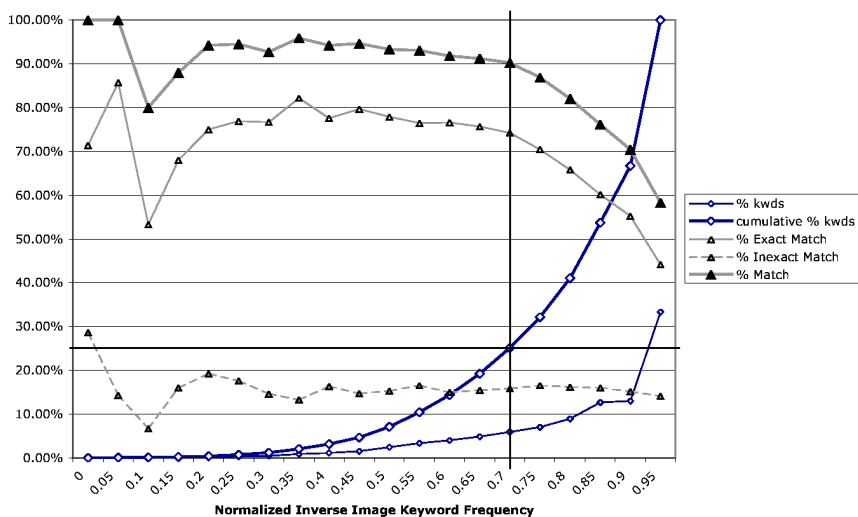


Fig. 5. The accuracy of the mapping between Corbis keywords and WordNet lemmas. The x axis is an inverse function of the frequency a keyword appears in the image collection. Keywords that appear most frequently have lower values on the x axis. The intersection of the lines $x=.725$, $y=25\%$ shows that the 25% most frequently used keywords are matched to lemmas with greater than 90% accuracy (75% exact and 15% inexact)

however, the user is able to rapidly choose the examples that are most relevant to their search. Fig. 6 shows a comparison between finding images using keywords and using concepts. Both queries begin from text that must be matched to a keyword, $T \Rightarrow K$, or lemma, $T \Rightarrow L$ (*lexical disambiguation*). To use concepts, we must determine which sense of the lemma is correct, $L \Rightarrow C$ (*semantic disambiguation*), then how to represent this concept in keywords, $C \Rightarrow K$ (*semantic disambiguation*).

Fortunately, many of the images are hierarchically annotated (presumably with a tool like WordNet) to aid in semantic disambiguation. For example, Fig. 7 shows the overlap between the keywords associated with the image shown earlier (on the left in bold) and the corresponding concept in the *Is-A* hierarchy for an Arabian horse in WordNet (on the right in bold).

This semantic redundancy in the annotation enables a user to find images when they search using general terms. However, hierarchical annotation by itself does not provide a structure for exploring the collection. On the other hand, using a general semantic model, the user would be able to navigate along the relationships defined in the model, but not necessarily instantiated in the collection. Our prototype uses both together, allowing the user transparent access to the structure of the model to navigate within the image collection. In the following section we define more formally how we create a semantic fisheye view using images, keywords and concepts.

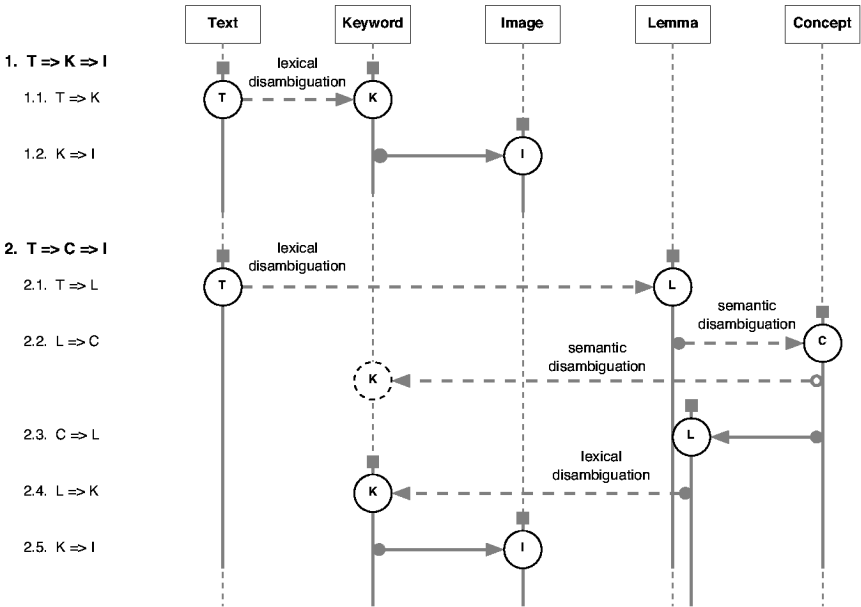


Fig. 6. A comparison of searching for images by keyword and searching by concept. Searching with concepts introduces the semantic ambiguity of selecting the correct sense of the lemma, $L \Rightarrow C$, and lexical ambiguity because of the incomplete match between lemmas and keywords, $L \Rightarrow K$

Corbis	WordNet
	
k1 : Heads	c1 : Arabian, Arab
k2 : Arabian horses	c2 : saddle horse, ...
k3 : Animals	c3 : horse, Equus caballus
k4 : Horses	c4 : equine, equid
k5 : Equus	c5 : odd-toed ungulate, ...
k6 : Odd-toed ungulate	c6 : ungulate, hoofed animal
k7 : Hoofed mammals	c7 : placental, ...
k8 : Herbivores	c8 : mammal
k9 : Mammals	c9 : vertebrate, ...
k10 : Vertebrates	c10 : chordate
k11 : Chordates	c11 : animal, ...

Fig. 7. The correspondence between the hierarchical annotation in Corbis, and the *Is-A* hierarchy in WordNet

4.6 Modeling “Focus + Context”

A semantic fisheye view is created by determining the degree of interest of each object in a collection with respect to the user’s current focus, as described earlier in equations 1 and 2. In our interface, we use the following general DOI equation:

$$DOI_{context}(x|fp = y) = API(x) - \sum_{i=1}^n w_i dist_i(x, y) . \quad (5)$$

The focus can be defined as a tuple composed of weighted vectors of keywords, images, lemmas, concepts, and previous foci:

$$f_i = \langle K_i^*, I_i^*, L_i^*, C_i^*, f_{n < i}^* \rangle . \quad (6)$$

In order to determine the distance between objects in different domains, we project the focus into the domain of interest. In addition to the unambiguous mappings from images to keywords, $I \rightarrow K^*$ (equation 4), and concepts to lemmas, $C \rightarrow L^*$ (equation 5), we define the following additional ambiguous transformations, where w is the probability of the match:

$$L_i \Rightarrow C_i^* = \{w_1 c_1, \dots, w_n c_n\} . \quad (7)$$

$$L_i \Rightarrow K_i^* = \{w_1 k_1, \dots, w_n k_n\} . \quad (8)$$

$$K_i \Rightarrow L_i^* = \{w_1 l_1, \dots, w_n l_n\} . \quad (9)$$

$$K_i \Rightarrow I_i^* = \{w_1 i_1, \dots, w_n i_n\} . \quad (10)$$

A fisheye view is created by determining the distance from the user's focus to the related objects in the collection using these equations. The probabilities are then represented visually as described in Table 1, and shown in Fig. 2.

5 Conclusions and Future Work

Opportunistic search, exploration and semantic discovery in large collections of information can greatly benefit from interfaces that are able to integrate relevant semantic models with interactive visualization. In this type of environment, a perfect match between the collection and the semantic model is not necessary for effective opportunistic search. A user of this type of environment is at the early, exploratory phase of search where it is important to gain a broad overview of the concepts and vocabulary of a collection.

We have presented a prototype environment for exploring an annotated image collection based on *semantic fisheye views*. This prototype integrates WordNet by deriving a mapping between the keywords used to annotate images and the lemmas associated with WordNet concepts. The user is then able to search for images based on the semantic relationships between concepts defined in WordNet. As the user changes their focus by navigating between images, keywords, and concepts, the *semantic fisheye view* modifies the representation of the objects in the collection to reflect the user's *degree of interest*. The visual *emphasis techniques* reduce the complexity of the display, and the fluid interaction allows the user to discover new domain concepts and relationships, and suggest future directions for search.

References

1. Bates, M.J.: The design of browsing and berrypicking techniques for the online search interface. *Online Review*, **13** (1989) 407–424
2. Bates, M.J.: Information Search Tactics. *Journal of the American Society for Information Science*, **30** (1979) 205–214
3. Bates, M.J.: Where should the person stop and the information search interface start? *Information Processing and Management*, **26** (1990) 575–591
4. Furnas, G.W. and S.J. Rauch: Considerations for Information Environments and the NaviQue Workspace. In: *Proceedings of the 3rd ACM International Conference on Digital Libraries (DL '98)*. (1998) 79–88
5. Pu, P. and P. Janecek: Visual interfaces for opportunistic information seeking. In: Stephanidis, C., Jacko, J. (eds): *Proceedings of the 10th International Conference on Human Computer Interaction (HCII '03)*. Crete, Greece: Lawrence Erlbaum Associates (2003) 1131–1135
6. Furnas, G.W.: Generalized Fisheye Views. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '86)*. (1986) 16–23
7. Rao, R. and S.K. Card: The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94)*. (1994) 318–322
8. Sarkar, M. and M.H. Brown: Graphical Fisheye Views of Graphs. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '92)*. (1992) 83–91
9. Janecek, P. and P. Pu: A Framework for Designing Fisheye Views to Support Multiple Semantic Contexts. In: *Proceedings of the ACM International Conference on Advanced Visual Interfaces (AVI '02)*. ACM Press: Trento, Italy. (1992) 51–58
10. Miller, G.A.: WordNet: a lexical database for English. *Communications of the ACM*, **38** (1995) 39–41

Ontology-Mediated Service Matching and Adaptation

Han-Kuan Yu, Tse-Ming Tsai, Yong-Fang Yang, and Wen-Shi Huang

Advanced e-Commerce Technology Laboratory Institute for Information Industry, Taipei,
Taiwan, R.O.C.

{hankuan, eric, yangyf, simon}@iii.org.tw

Abstract. The ontology-mediated modeling of matching service requirements with providers, involves parsing a user's intention, providing right task flow, and adapting the environment change. This research introduces a practical approach to satisfying users with high levels of service integration beyond a simple interaction. After the semantics of Web Services interfaces are registered to platform, for which a task flow is developed, three matching comparators - context-based, signature-based, and constraint-based matching - are applied to analyze the task flow that best suits the user's intentions, parsing by plaintext input. For practical purposes, the adaptation process of task flow and ontology is modeled, and the user could approve or discard the proposed suggestions.

1 Introduction

The ubiquity of the Internet and Web standards, such as TCP/IP, HTTP and HTML, makes the global village a reality in cyberspace. XML (Extensible Markup Language) has become the preferred alternative for exchanging messages among heterogeneous systems. Additionally, emerging schema and semantic standards, such as DTD, XML schema, RDF, DAML+OIL, OWL and others, have enabled the description of meta data and the development of ontology for semantic expression.

XML-based Web Services standards, such as SOAP, WSDL and UDDI, all of which promise further easy integration with open Internet protocols, enable machines to communicate with each other just like humans. As well as those to extend accessibility through the Web, sustained efforts are being made to take advantage of the mutual understanding of a range of Web contents, which are conceptualized as the Semantic Web.

Ubiquitous services may be possible in the future. For an end user, more available services may imply more freedom, or more chaos. Users should be served with a well integrated UI rather than confused by granularly separated operations. The most natural and intuitive UI should take a user's plain-text sentence as a system input and then push services that meet the user's requirement.

Using a mediating ontology, user intention and candidate process both can be modeled in the same representation, such that semantic matching can be made increasingly precise and effective. Modeling the entire matching procedure - parsing

user's intentions, providing the right task flow, and adapting the environment change - this work introduces a practical approach to satisfying users with high levels of service integration beyond a simple interaction.

2 Related Work

2.1 Parsing the User's Intention

Natural Language Processing (NLP) must be effectively used to understand the plain-text intention of an end user. Unlike common information retrieval techniques that extract keywords from a text data source, NLP seeks to understand the semantic meanings of sentences.

SCISOR (System for Conceptual Information Summarization, Organization and Retrieval) is a prototype information retrieval system of the future [8]. SCISOR reads news stories about mergers and acquisitions from a newswire source, extracts information from the stories, and answers questions in English that pertain to what it has read. Input questions are parsed using the same understanding mechanism as is used for the input stories. SCISOR continuously accumulates knowledge of the news, and can answer questions automatically.

Additionally, named-entity detection is crucial for extracting a user's intention. Named entities such as people's names, dates, time, monetary amounts and proper nouns, are all important semantic units in a user statement. Some research in this area has focused on general and domain-specific heuristic rules to locate named entities from the plain-text content [3].

2.2 Semantic Matching

Keyword-based and frame-based matching approaches are widely used to identify services or contents that meet a user's requirements. Additionally, semantic matching considers the semantic meaning of requests and resources [5].

LARKS (Language for Advertisement and Request for Knowledge Sharing), developed at the Intelligent Software Agents Laboratory of Carnegie Mellon University, was introduced to perform dynamic service matchmaking among heterogeneous software agents on the Internet. LARKS performs not only syntactic and semantic matching, but also compares specifications of ontology using ITL (Information Terminological Language), a concept language. LARKS involves three types of general agents - Provider Agent, Request Agent and Matchmaker Agent. Matchmaking is a process by which a Matchmaker agent finds the service providers to meet the ITL requirements of requestors. An advertisement is a document that describes the provider's functional capabilities, conditions and effects, according to which the matchmaker agents match service providers to the requirements of requestors [4].

DAML-S is a semantic markup language based on the DAML for automating Web services. The DAML-S models the capability profile, the process logic, and the invo-

cation (grounding) information of each service. The DAML-S is an internet-suitable solution, however it might be too expensive for most intranets.

2.3 Ontology Adaptation

Intelligent applications can use shared ontology to identify semantic meaning and then solve a problem or infer an outcome. Ontology is applied as a theory of content that pertains to types of objects, their properties, and the possible relationships among objects in a specified domain of knowledge [2].

An ontology-based system uses ontology to specify explicitly a business domain. However, dynamic changes in the business environment imply changes in the conceptualization of a business domain that are reflected in the underlying domain ontologies [7]. Ontology adaptation involves continuous reorganization of the ontology, which may induce semantic inconsistency and thus reduce the accuracy of the systems. Stojanovic et al. proposed an approach that systematically analyzes the cause and the consequences of such changes and ensure the consistency of the ontology and related ontologies [1, 6].

3 Design Rationale and Architecture

Currently, the user, who performs the Web Services composition task, must specify the Web Services manually, and ensures that all information needed to compose the task is well defined. A high-level language for describing Web Services process logic must be specified in the design phase to support automatic composition of Web Services. Business Process Execution Language for Web Services (BPEL4WS) [9], an XML-based language for describing workflow processes, are currently being developed for this purpose. To an extent, a software agent can dynamically perform selection, composition and interoperation of Web Services to accomplish users' tasks.

A platform must be supported by flow modeling and flow adaptation to automate task flow automation. As users execute the proposed flow, or any other alternatives, their behavior information can be logged and monitored to the adaptation function. Figure 1 depicts the design rationale of the platform. When a task description, for example, a document, an e-mail or a sentence, is inputted to the platform, the Intention Parsing module transforms it to a machine-understandable format. The matching procedure is then executed to send a suitable task flow to the user for handling the requirement. The mistaken matchmaking events on the platform are treated by the adaptation procedure to adapt the office environment and the behavior of the user.

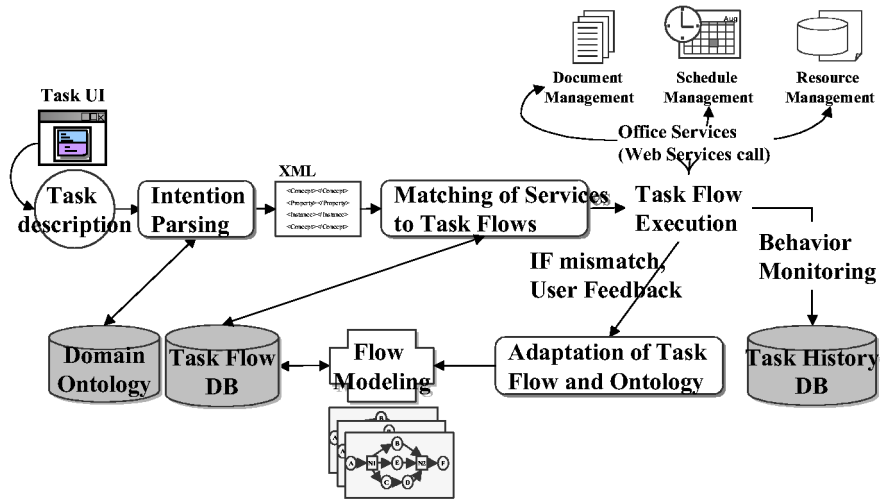


Fig. 1. Complete matching procedure from intention to task flow

4 Intention Parsing

When a plain-text task requirement is sent to the proposed platform, the agent translates task semantics into process actions and automates a series of services to meet associated requirements. For translation purposes, the agent must parse the requirement into a machine-understandable format and then match it to a suitable task flow. Domain ontology is applied here to link user intention to machine understanding to perform a task. AutoTag, an NLP tool that was developed by the CKIP group of the Academia Sinica in Taiwan, for word segmentation, preprocesses every incoming requirement sentence. After the agent obtains a sequence of words, it identifies the concepts, concept properties and concept instances by comparing them with those defined in the domain ontology. Depending on the identified outputs, the agent organizes the Semantic Patterns to extract more meaningful semantics associated with the requirements. A Semantic Pattern is a serial combination of a subject instance, a subject predicate (property) and an object instance. The output instances and predicates are assembled into possible Semantic Patterns by applying some merging and pruning rules. In the most cases, the recognition of instances dominates whether the outcome Semantic Patterns are good or bad. In the proposed design, the agent obtains knowledge that pertains to instance identification from the General Instance Rule Base and the Domain Instance Rule Base. The General Instance Rule Base includes rules for named-entity detection (such as people, things, time, locations). The Domain Instance Rule Base, defined by domain experts, contains rules that can identify proper nouns associated with domains. In this phase, the agent transforms requirements to the Semantic Patterns, and records the patterns in a well-structured DAML format. Figure 2 presents the mechanism.

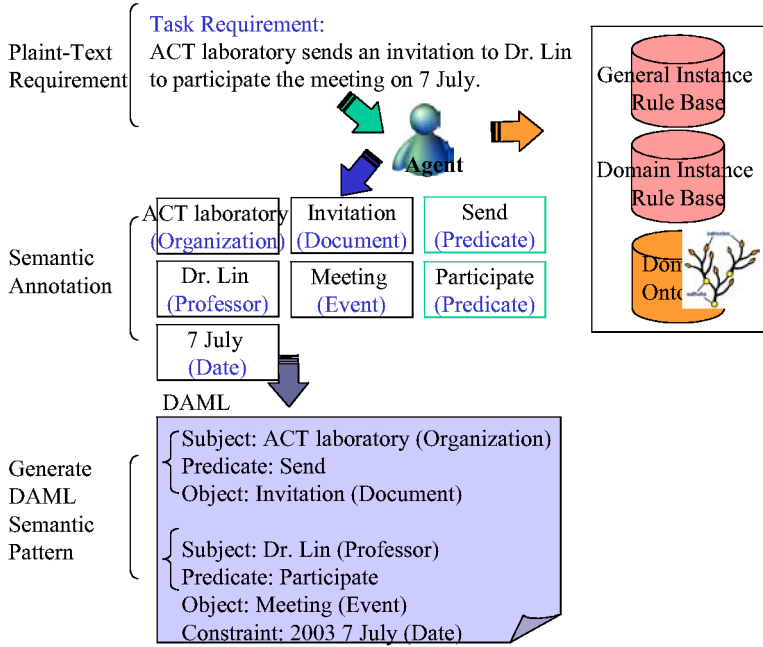


Fig. 2. Formulating free-text requirements in structured DAML form

For example, a task is described as, "ACT laboratory sends an invitation to Dr. Lin to participate in a meeting on 7 July". Each recognized term is firstly annotated with a semantic label by applying Domain Ontology and instance rules. In this example, "ACT laboratory", "Invitation", "Dr Lin", "Meeting" and "7 July" are recognized as semantic instance objects, and "Send", "Participate" are addressed to predicates. Two Semantic Patterns, {"ACT laboratory-Send-Invitation"} and {"Dr. Lin-Participate-Meeting-7 July"}, are extracted into DAML form for performing further matchmaking.

5 Matching of Services to Task Flows

In the intention parsing phase, a task requirement is translated into a machine-understandable format. The task flows on the platform also express their mission to perform matching and thus automatically connect the request to the suitable flow. Two types of registration mechanism are considered herein for conveying the semantics of the task flows and the office services to the platform.

5.1 Registering the Office Services

On the platform, each office service must publish a WSDL document, the standard entity for describing a Web Service, and register a Service Profile that associates the semantic meaning of the service with each method and input/output parameter. When the semantics of each service are registered at the platform, developers must associate each method and input/output parameter with the corresponding concept or property in the shared ontology. Consider the example shown in Fig. 3, in which a Web Service method Oper2: bookMeetingRoom requires five input parameters - userID, Date, startDate, endDate and meetingRoom - and generates a result represented by an output parameter, recordID described in WSDL. The corresponding support of the Domain Ontology is the userID parameter is an EmpID property of a BOOKING_RECORD concept; it also refers to the Id property of the EMPLOYEE concept. Similarly, the startDate parameter is the STime property of the BOOKING_RECORD concept, whereas the range of STime property is START_TIME. When the service provider publishes the service to the platform, he or she must provide the semantic linkage among each service I/O parameter to complete the registration.

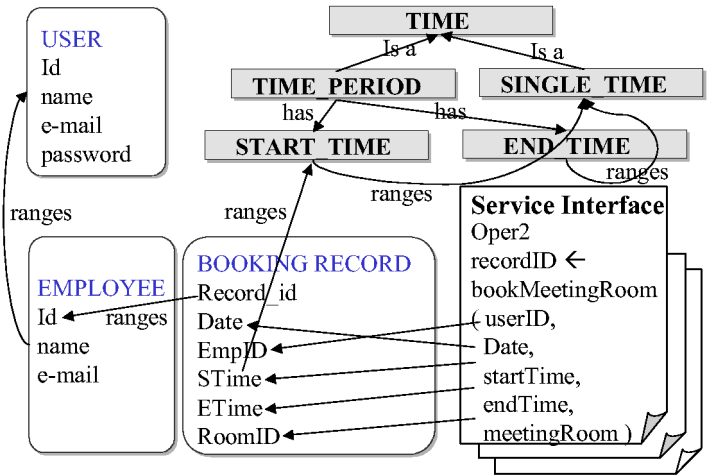


Fig. 3. Registering the office services

5.2 Registering the Task Flows

The composed process services, so-called complex services in the Web Services field, can be importantly involved in matching and pushing the right and complete process for a specific task, which is called a task flow. As with the office service, the semantic meaning of the task flow is expressed via a registered Service Profile. A Service Profile of the task describes the context, Concept Patterns, and Instance Patterns of the task. The task context expresses the type of job in the office, and can be inferred automatically from the Domain Task Rule Base or by manual selection. The Semantic

Patterns of a task represent the signatures of the task description, and they accumulate automatically into the Service Profile. Additionally, the eXFlow profile is another crucial document for describing the plan for executing the task flow. On the platform, the task flow designer formulates the process of the office task by the Visualization tool, Process Definition Designer, and registers the output eXFlow profile in the Task Flow DB.

Identical or similar methods and their I/O parameters in different Web Services interfaces have often been argued to have different meanings because of various naming conventions and expressions. Domain Ontology facilitates communication among task requirements, office services and task flows through a shared knowledge pool, enabling automatic task processing.

5.3 Matching Procedure

When a user's task has been extracted as the Semantic Patterns, the matchmaking mechanism is responsible for finding a suitable task flow to meet the user's requirements. A structured intention form is understandable by an agent and can be compared with the capability descriptions of the task flows registered on the platform. Three matching comparators, Context-based, Signature-based and Constraint-based, are considered to measure the semantic distances between requirements and the capabilities of the office services.

5.3.1 Context-Based Matching

Context-based matching checks the matching between two specifications within the same context. In the proposed design, a DAML requirement profile has a <Context/> tag that defines the type of requirement to narrow the search space. Any two contexts are the same if the context distances between two specifications do not exceed a specified threshold. In such a case, the requirement and the service flow are in the same task context. In practical usage, each context of a task type is linked to an ontological concept and is logged in the Service Profile. As shown in Fig. 4, the meeting record and official document are different document types in the office, and are usually processed by completely different procedures. In the 'Intention Parsing' stages, the agent must distinguish types of requirement sources according to Domain Task Rule Base, and sign the requirement context. The rules in knowledge-based systems must be represented in a formal structure so that they can be processed by inference engines. The identified implicit rules are transformed into the formal rule structure of RSML. A rule is introduced herein to describe the signatures of the title and the URI of an official document in our office.

When the HTML title of a requirement document is 'III Official Document:*' and the document comes from the URI 'http://iii.org.tw/officialDocument/*' or 'http://140.92.140.1/officialDocument/*', a document with the context 'http://.../sota#OfficialDocument' is established. The context type of a requirement is used as the flag in the preliminary analysis involved in the matching procedure.

<RSML>

```
<Rule>
  <RuleTitle>Rule of Official Document</RuleTitle>
  <IF>
    <AND>
      <HTMLTitle>III Official Document:*</HTMLTitle>
      <OR>
        <DocumentURI>http://iii.org.tw/officialDocument/*</DocumentURI>
        <DocumentURI>http://140.92.10.1/officialDocument/*</DocumentURI>
      </OR>
    </AND>
  </IF>
  <THEN>
    <Result Action="SetContext">
      <Context>http://.../sota#OfficialDocument</Context>
    </Result>
  </THEN>
</Rule>
</RSML>
```

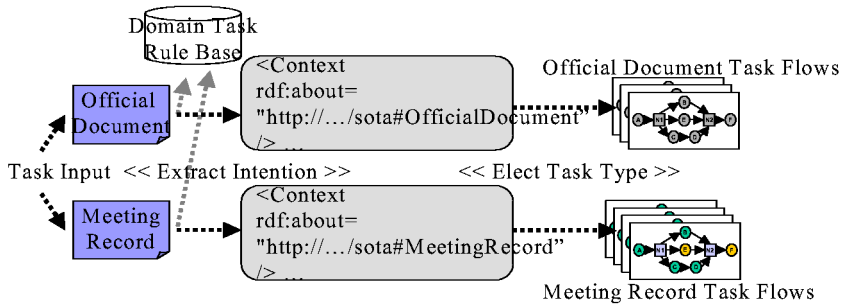


Fig. 4. Context-based matching

5.3.2 Signature-Based Matching

Signature-based matching is used to consider the detailed semantics of the task requirement. Two Semantic Patterns - Concept Pattern and Instance Pattern - are defined to represent the signatures from plain-text description. The forms of the two patterns are as follows.

1. Concept Pattern: 'subject concept'-'predicate'-'object concept'
2. Instance Pattern: 'subject instance'-'predicate'-'object instance'

The task requirement is converted into the above two Semantic Patterns in the 'Intention Parsing' stage, and all patterns are logged in the Requirement Profile. Signature-based matching compares Concept Patterns and separately Instance Patterns to measure the distance between requirement signatures and the past task signatures. When two Concept Patterns are matched, two description sentences are similar, and Instance Patterns are then compared. The comparison of Instance Patterns computes

distances between pairs of subject instances and pairs of object instances. It measures instance data to determine whether the subject (or object) data are the same.

As shown in Fig. 5, when the platform receives a new task description:

'ACT laboratory sends an invitation to ...'

Platform extracts the signatures into the Semantic Patterns:

Concept Pattern: Organization-Send-Document

Instance Pattern: ACT Laboratory-Send-Invitation

Platform compares the Semantic Patterns of the requirement with those of the past tasks, and picks out the flows of similar tasks. Signature-based matching considers the sentence patterns and instance data of the task requirement. Precisely determining similar task descriptions and obtaining candidate task flows from the experience of past tasks is useful.

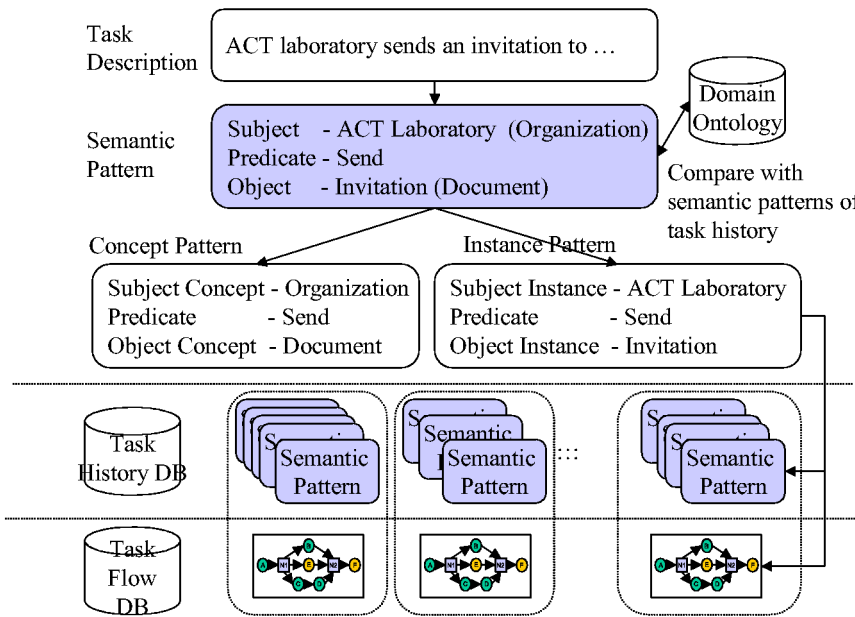


Fig. 5. Signature-based matching

5.3.3 Constraint-Based Matching

Constraint-based matching evaluates the fitness of inputs, outputs and constraints related to the request and the capability advertisements of services. The user's output requirements/constraints and the services' input constraints are set as necessary in the matching procedure to assure the completion of the request. Constraint-based matching is typically used for allocating resources, for example, when booking equipment or arranging schedules. In real-world cases, precisely extracting constraint intentions from an unstructured plain-text description is difficult. The proposed design uses pre-defined forms that consist of constraint fields of some specific tasks for collecting the

users' constraints and output intentions. Figure 6 presents the task of reserving 'Meeting room B' for a meeting. Platform extracts input data instances ('890117', '2003/07/23', '09:00') and output requirements / constraints (Room_B) on this requirement, and performs constraint-based matching to evaluate suitable task flows. At the bottom of Fig. 6, all I/O parameters and constraints of the task flows link to the conceptual properties of the domain ontology to elucidate their semantic meaning. Constraint-based matching validates the fitness of I/O parameters and constraints of the services in a task flow by comparing semantic linkages. Constraint-based matching aims to measure whether the inputs and outputs of the task flow can meet the requirements of the task.

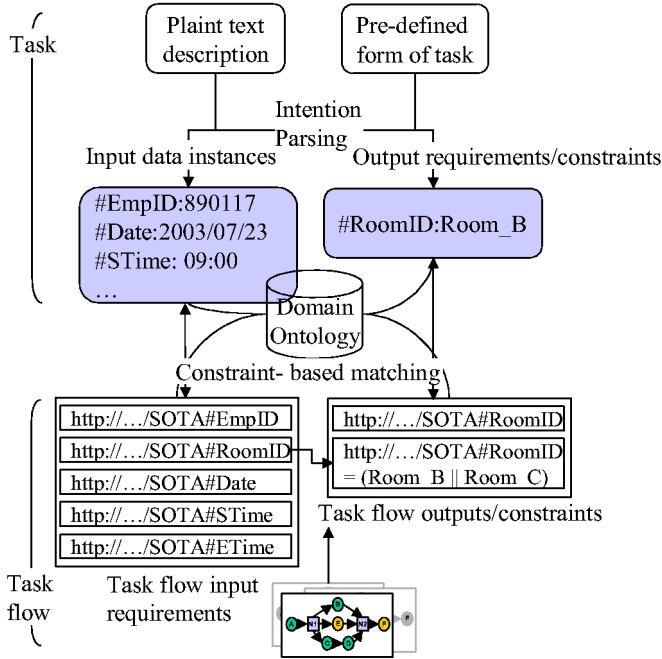


Fig. 6. Constraint-based matching

Each matching procedure M_i is weighted by a constant W_{M_i} that multiplies the matching score $S_{M_i}(\text{requirement}, \text{taskflow}_i)$ to determine the matching distance between the new requirement and taskflow_i of M_i :

$$D_{M_i}(\text{requirement}, \text{taskflow}_i) = W_{M_i} * S_{M_i}(\text{requirement}, \text{taskflow}_i)$$

The total distance between the new requirement and taskflow_i outputs from matching procedure will be,

$$D(\text{requirement}, \text{taskflow}_i) = D_{\text{context}} + D_{\text{signature}} + D_{\text{constraint}}$$

The output scores from the three matching procedures of each task flow are summed, and the flow with the best total is selected for handling the input requirement. The matching procedure is key to achieving the goal of automatic task execution.

6 Adaptation of Task Flow and Ontology

The matching procedure must incorporate an adaptation mechanism to prevent ageing. When a user executes a task flow that is inconsistent with the recommended flow of the matching procedure, the matchmaking is defined as mistaken. Such a mistaken matchmaking event triggers an adaptation procedure on the platform. Two types of adaptation procedures - task flow adaptation and ontology adaptation - are implemented herein to maintain the precision and recall rates of the matching procedure.

6.1 Task Flow Adaptation

On the platform, every step in the processing of the tasks is recorded in the Task History DB. When the mistaken matchmaking is detected, the platform checks the execution log of this task. If the processing steps are the same as those of a current task flow on the platform, then ontology adaptation is performed. However, if the processing steps differ from all registered task flows, then the platform must judge whether to form a new task flow for the processing steps. The platform performs a statistical calculation to measure the utility rate of the processing steps. If the confidence level of H_0 : the utility rate of the new task flow $> k'$ exceeds a specified confidence threshold, then the user will be prompted to confirm the formation of this new task flow. As shown in Fig. 7, the adaptation model of task flow automatically generates Service Profile (task context, task Semantic Patterns and semantic linkage among service I/O parameters) and eXFlow Profile (the service flow description of the proposed workflow engine) for the new task flow.

6.2 Ontology Adaptation

When task intention cannot be interpreted correctly, a mistaken matchmaking will arise. The definitions of concepts, concept properties and concept instances in Domain Ontology dominate the recognition performance of the Semantic Pattern. If the requirement sentence consists of critical vocabulary unexplained by the platform, then the ontology adaptation is performed. The right-hand side of Fig. 7 presents the procedure. Platform separates words from unknown sentences, and analyzes the part of speech to determine possible Semantic Patterns. All the verified new knowledge is added to the Domain Ontology, to accumulate new capabilities for handling the upcoming requirement.

The change of the office environment might cause the matching procedure imprecisely. The adaptation procedure is considered handling the events of mistaken matchmaking via a semi-automatic mechanism. We aim to enable the ability of adaptation for the matching procedure on the platform and build the reliable environment for the automatic task processing.

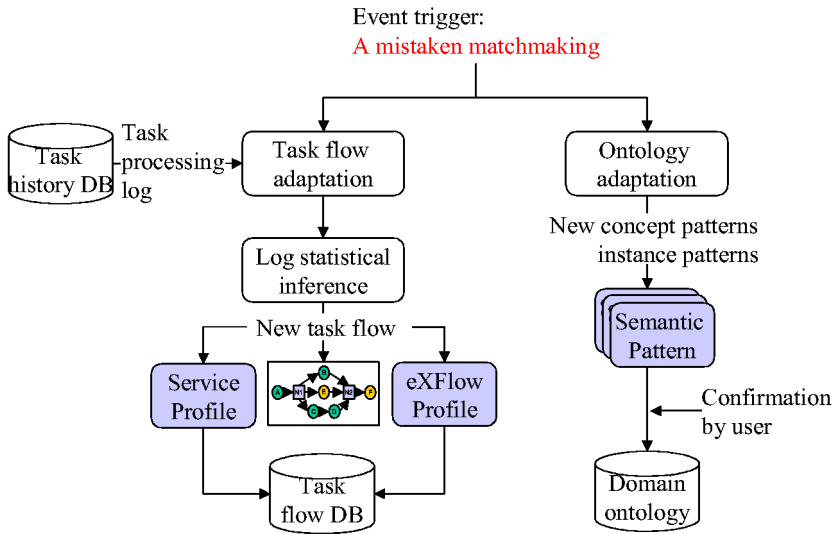


Fig. 7. Adaptation Model

7 Conclusion

Web Services enable the remote invocation of heterogeneous systems, and support a perfect means of integrating diverse systems into a task-oriented process. Ontology opens up the possibility of machine-readability and precise understanding among different parties. Effectively modeling the semantics of domain ontology and process services, this work provides a proactive and adaptive approach to matching the user intentions to right task flow parsed by plaintext input. Platform aims for facilitating users to deal with complex tasks at the most compressed interaction.

Acknowledgement. This research was supported by the III Innovative and Prospective Technologies project of Institute for Information Industry and sponsored by MOEA ,R.O.C

References

1. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "Ontologies for Enterprise Knowledge Management," IEEE Intelligent Systems, 2003.
2. B. Chandrasekaran, J.R. Josephson, V.R. Benjamins, "What Are Ontologies, and Why Do We Need Them?" IEEE Intelligent Systems, 20–26, Jan/Feb 1999.

3. Hsin-Hsi Chen and Guo-Wei Bian, 1998, "White Page Construction from Web Pages for Finding People on the Internet", *Computational Linguistics and Chinese Language Processing*, vol. 3, no. 1, 75–100, February 1998.
4. Katia Sycara ,Seth Widoff ,Matthias Klusch and Jianguo Lu, *Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace*, *Autonomous Agents and Multi-Agent Systems*, 5, 173–203, 2002.
5. Lei Li and Ian Horrocks, "A Software Framework For matchmaking Based on Semantic Web Technology," *Proceedings of the twelfth international conference on World Wide Web 2003*, 331–339, 2003.
6. L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic, "User-driven Ontology Evolution Management," In *Proceeding of 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW-2002)*, Siguenza, Spain, October 2002.
7. N. Stojanovic, L. Stojanovic, "Evolution in the ontology-based knowledge management system," *Proceedings of the European Conference on Information Systems – ECIS 2002*, Gdańsk, Poland, 2002.
8. P. S. Jacob, L. F. Rau, "Natural Language Techniques for Intelligent Information Retrieval", *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, May 1988.
9. Tony Andrews, et al., "Specification: Business Process Execution Language for Web Services Version 1.1," 05 May 2003, <http://www-106.ibm.com/developerworks/library/ws-bpel/>

The Role of Vague Categories in Semantic and Adaptive Web Interfaces

Miguel-Ángel Sicilia

Computer Science Department, Carlos III University,
Av. Universidad, 30 – 28911 Leganés (Madrid), Spain

`msicilia@inf.uc3m.es`

Abstract. Current Semantic Web technologies provide a logic-based framework for the development of advanced, adaptive applications based on ontologies. But the experience in using them has shown that, in some cases, it would be convenient to extend its logic support to handle vagueness and imprecision in some way. In this paper, the role of vagueness in the description of Web user interface characteristics is addressed, from the viewpoint of the design of adaptive behaviors that are connected to such descriptions. Concretely, vague descriptions combined with quantified fuzzy rules and flexible connectors are described, and their usefulness is illustrated through preference modeling, filtering and adaptive linking scenarios.

1 Introduction

The vision of a Semantic Web [4] has recently attracted the attention of researchers coming from various disciplines. The essential element of that vision is the provision of meta-information for Web resources, in a machine-understandable form, so that it can be shared and processed by automated tools, thus enabling a new generation of ‘intelligent’ applications. Consequently, both (a) a *lingua franca* for annotation, and also (b) a number of consensual terminologies are required to guarantee a common understanding of such meta-information. *Description Logics* (DL) [2] and *formal ontologies* expressed using them [11] have been proposed as the response for requirements (a) and (b) respectively [3]. In the last decade, a substantial amount of theoretical and applied research regarding DL has been carried out [2], resulting in logics bearing a good compromise between expressive power and computational complexity. Nonetheless, since the publication of McCarthy’s seminal papers on epistemological problems — see, for example, [17] —, any problem of artificial intelligence (AI) is considered to comprise an epistemological part, and a heuristic part. The former is concerned with the representation of facts about the world as are available to us, while the latter is concerned with the algorithms required to make AI computations feasible. Therefore, despite the heuristic appropriateness of current DLs to many common problems, it still remains necessary an inquiry about its epistemological adequacy for the kinds of knowledge that should be encoded in Semantic Web applications.

Here we are mainly concerned with the epistemological adequacy of Semantic Web applications, with regards to vagueness as a human categorization characteristic [18]. More concretely, previous experiences [8,16,19,9] have lead us to consider vague categories as an essential problem component in user modeling and adaptation regarding characteristics of human's perceptions of Web user interfaces. These aspects include descriptions of Web user interface (UI) elements — like for example, sizes, density, granularity and link structure —, not directly related to the content (i.e. the meaning) of the page. This characteristic makes UI descriptions domain-independent, and in consequence, of a wide applicability, orthogonal and complementary to annotations regarding content description or authorship. In addition, these UI descriptions are important both for the representation of user models and to express adaptation rules. For example, user preferences regarding granularity may be expressed as “user U prefers long descriptions”, where *long* represents a fuzzy subset defined on the length (measured in words, for example) of content items, and a rule like “if student is novice then provide him with simple explanations” enables adapting the density of explanation items using the imprecise *simple* category. It may be argued that collapsing these kind of vague concepts into discrete enumerations (e.g. {very long, long, medium, short, very short} for Web pages) could solve the problem from a practical perspective. But previous research on membership elicitation of simple Web characteristics [9] provides evidence against such claim, since elicited fuzzy subsets for a given category are neither regular in function shape, nor equally spaced in the domain of definition. In addition, some adaptive hypermedia technologies [6] are adequate for partial truth-transfer inferencing. For example, one of the most typical link navigation adaptive technologies is link annotation. If the attribute of the link to be adapted is of a variable nature like font size (or font color), a rule with a consequent part like “change font-size of link L to large” may produce different “large” fonts according to the degree of activation of some fuzzy antecedent. The concept of personalized fuzzy links [21] are a different formulation for that concept of imprecise adaptiveness.

Previous research have addressed the integration of vague concepts into crisp description logics [25,26,12], resulting in what can be called fuzzy description logics (fDL). Retrieval of multimedia objects [24] and electronic commerce [12] have been mentioned as application areas for fDLs. In this paper, we describe vagueness in UI descriptions as a component of a Semantic Web application, with the objective of providing a framework for adaptation. A number of useful imprecision handling mechanisms that can be integrated with DL are described, without the aim of studying their formal logical properties, that are left to future work. The motivation for those mechanisms is of a practical nature, and consequently, concrete examples we have faced with are provided as well.

The rest of this paper is structured as follows. In Section 2, a general framework for the vague semantic description of adaptive Web user interfaces is sketched. Section 3 illustrates the applicability of such models for concrete adaptive techniques. Finally, conclusions and future research directions are provided in Section 4.

2 Vagueness in Adaptive, Semantic Web Interface Models

The concepts of Adaptive Web [7] and Semantic Web can be considered as two perspectives of the same vision of a prospective Web. While the former one focus on tailoring a hypermedia space to the heterogeneity of user goals, knowledge and interests, the latter one takes a broader view, concentrating on the availability of shared knowledge representations and a common technological support for the development of advanced applications — including adaptive ones—. From here on, we'll follow the assumption that both research areas will converge in the future, and therefore, DLs, perhaps extended or combined with other paradigms, will form the basis of Adaptive-Semantic Web sites. In addition, we take the $\mathcal{ALC}_{FH}$ fuzzy description logic described in [12] (which in turn is an extension of the Fuzzy- $\mathcal{ALC}$ defined in [25]) as a point of departure for the discussion of practical aspects of vague semantics in user interface descriptions.

In $\mathcal{ALC}_{FH}$, concepts are expressions that describe, by means of roles, the properties of a set of individuals. Given that C and D are concepts, M is a concept modifier, and R is a role, we have the following syntax rules:

$C, D \longrightarrow$	$\top$ (top concept)
	$\perp$ (bottom concept)
	A (primitive concept)
	R (primitive role)
	$C \sqcap D$ (concept conjunction)
	$C \sqcup D$ (concept disjunction)
	$\neg C$ (concept negation)
	MA (concept modification)
	$\forall R.C$ (universal quantification)
	$\exists R.C$ (existential quantification)

The semantics of $\mathcal{ALC}_{FH}$ are based on the following interpretations, where η_M is a membership modifier (this category is intended to model fuzzy hedges like *VERY* or *FAIRLY*):

$$\begin{aligned}
 \top^{\mathcal{I}}(d) &= 1 \quad \forall d \in \Delta^{\mathcal{I}} \\
 \perp^{\mathcal{I}}(d) &= 0 \quad \forall d \in \Delta^{\mathcal{I}} \\
 A^{\mathcal{I}} &: \Delta^{\mathcal{I}} \rightarrow [0, 1] \\
 R^{\mathcal{I}} &: \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow [0, 1] \\
 (C \sqcap D)^{\mathcal{I}}(d) &= \min\{C^{\mathcal{I}}(d), D^{\mathcal{I}}(d)\} \\
 (C \sqcup D)^{\mathcal{I}}(d) &= \max\{C^{\mathcal{I}}(d), D^{\mathcal{I}}(d)\} \\
 (\neg C)^{\mathcal{I}}(d) &= 1 - C^{\mathcal{I}}(d) \\
 (MA)^{\mathcal{I}}(d) &= \eta_M(A^{\mathcal{I}}(d)) \\
 (\forall R.C)^{\mathcal{I}}(d) &= \inf_{d' \in \Delta^{\mathcal{I}}} \{\max\{1 - R^{\mathcal{I}}(d, d'), C^{\mathcal{I}}(d')\}\} \\
 (\exists R.C)^{\mathcal{I}}(d) &= \sup_{d' \in \Delta^{\mathcal{I}}} \{\min\{R^{\mathcal{I}}(d, d'), C^{\mathcal{I}}(d')\}\}
 \end{aligned}$$

A (fuzzy) knowledge base Σ is the union of a finite set of fuzzy assertions Σ_A (often called “A-box”) and a finite set of fuzzy terminological axioms Σ_T (often called “T-box”). Terminological axioms are either fuzzy specializations

$A \lesssim B$ or fuzzy concept definitions $A \doteq C$. Fuzzy assertions take the forms summarized in expression (1), where $n \in [0, 1)$. For example, $\langle u : VIP \geq 0.8 \rangle$ may be interpreted as “user u belongs to the VIP user group to an extent greater or equal than 80%”, and $\langle (u, p) : interestedIn \leq 0.3 \rangle$ may be interpreted as “the interest of user u in product p is lower or equal than 30%”.

$$\left\langle \left\{ \begin{array}{l} a : C \\ (a, b) : R \end{array} \right\} \left\{ \begin{array}{l} > \\ < \\ \leq \\ \geq \\ = \end{array} \right\} n \right\rangle \quad (1)$$

Given the above definitions, the model of an adaptive hypermedia application can be logically described by means of $\mathcal{ALC}_{FH}$ as a knowledge base including the main elements that comprise the common architecture of such systems [27], extended to cope with imprecision in its principal constructs [22]. The main components of such architecture for a given system $\mathcal{AH}$ can be summarized in the four elements denoted in (2).

$$\mathcal{AH} = \prec HM, DM, UM, AM \succ \quad (2)$$

The hypermedia model (HM) describes the structure of the hypermedia application (Web applications are a concrete form of them), including contents, nodes and links. The domain model (DM) describes the vocabulary that is used for each concrete application, for example, an online bookstore will provide descriptions for books or CDs, while a educational Web application will have a DM including the concepts that are intended to be taught. The user model contains the description of the users and user categories of the application. Finally, the adaptation model — which is of a different nature to previous one — contains the description of the adaptive behaviors that must be carried out, in terms of the other models. It should be noted that expression (2) is only significant for ease of understanding, since the four models are actually tightly integrated. For example, preferences about books are a relationship traversing DM and UM, and adaptive behaviors in most cases tailor elements in the HM to information in UM and possibly also in DM. All the (sub-)models in (2) — with the possible exception of AH, that usually includes also rules or procedural formalism — can be described by a $\mathcal{ALC}_{FH}$ knowledge base $\Sigma_{\mathcal{AH}}$.

In the rest of this section, three concrete manifestations of vagueness that arise in the just described Adaptive-Semantic Web Model are described. For brevity’s sake, we’ll describe the concepts or relationships pertaining to the above models as they appear in the examples provided below. It should be noted also that, although the focus of this paper is on imprecise descriptions of user interfaces, these manifestations also apply to many cases of content description.

2.1 Vague Facts as Automated Assertions

Asserting facts according to the syntax in expression (1) may be unpractical in many situations. Membership functions of vague categories are a way to express

fuzzy degrees of belonging that can be used instead. These membership functions can be obtained from experts as approximations, or they can be obtained directly from users, by mean of existing membership elicitation techniques [5]. In the context of Web applications, a number of factors influencing the overall usability of the page have been studied [14], including amount of text, number and type of links and images, color use and page length. These characteristics can be modeled from user studies as collections of linguistic labels, as described in [9].

Given that membership functions are defined on a concrete (crisp) type of elements¹, we can state that the assertion of an instance may entail the immediate generation² of k other assertions belonging to predefined vague categories $c_1, c_2, \dots, c_k$ defined by their respective membership functions $\mu_{c_1}, \mu_{c_2}, \dots, \mu_{c_k}$. Each μ_j will have as domain of definition one or several discrete or continuous characteristics of the given type of instance B , that is, they will take the form $\mu_{c_j} : f^1(x) \times f^2(x) \cdots \times f^z(x) \rightarrow [0, 1]$ with $x : C$ and being each $f^j(x)$ a function transforming one or several primitive-type characteristics (e.g. relations from C to *Integer*) of a element belonging to C . For example, Figure 1 shows an approximation (using gaussian functions $g(x) = e^{-(\frac{x-c}{\sigma})^2}$) for the resulting membership functions *short*, *medium*, *long* and *very-long* (from left to right) described in [9]. These functions express the human understanding of the linguistic characterization of Web page length categories, for example, $\mu_{LongPage} : f^l(x) \rightarrow [0, 1]$ given that $x : WebPage$ and $f^l : C \rightarrow \mathbb{N}$ (defined on the domain of lengths in pixels).

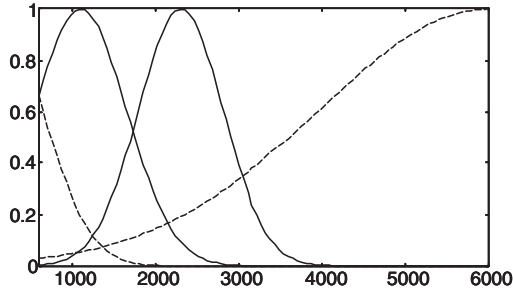


Fig. 1. Membership functions related to Web page length

This way, the assertion of an instance $\langle u : WebPage = 1 \rangle$ will produce four automatic assertions corresponding to each of the functions in Figure 1 (like, for example, $\langle u : Long = 0.6 \rangle$). Many user interface related descriptions — including granularity, sizes, semantic density and number of links or images — should be describe this way, since they qualify as imprecise human cognitive categories

¹ This may not be the case if we consider higher-order fuzzy sets, but we'll not deal with them here

² Of course, heuristic adequacy may preclude this *eager* approach, resulting in *lazy* asserting behaviors.

according to [18]. These functions may also be n -dimensional, for example, a category $\mu_{GTF} : f^{\#w}(x) \times f^{\#cwc}(x) \rightarrow [0, 1]$ may map word counts and column word counts to a degree of belonging to category of “well text-formatted” Web pages.

From the viewpoint of $\mathcal{ALC}_{FH}$, these automatic assertions do not entail semantic changes, but enable a nice alternative to the built-in hedges η_M described above. This alternative is that of introducing an explicit term M_C instead of introducing C , and relying on the logic-based interpretation of modifier M . In some cases, this enables being consistent with existing elicitation studies that question the validity of some “classical” hedges. For example, the *VERY* hedge is usually represented simply as $\mu_{VERY-A}(x) = \mu_A(x)^2$, while several experimental studies (summarized in [5]) report a shift in the membership function.

2.2 Integration of Fuzzy Rules with Quantifiers

Rules can be combined with description logics enabling forms of hybrid reasoning [1]. Conventional fuzzy rules and well-known fuzzy inferencing mechanisms can be integrated with $\mathcal{ALC}_{FH}$ in a straightforward manner. But fuzzy quantified rules entail a consideration for the cardinality of concepts that is not realized in interpretations of DLs as the above described $\mathcal{I}$. A fuzzy quantified rule can be denoted syntactically in two basic forms, absolute and relative, expressed respectively in (3) and (4). Absolute quantification refers to a single concept like in “**many** long pages”, while relative quantification represent a proportion like in “**most** long pages are dense”. Zadeh’ σ -count operator [29] is one of the possible operationalizations [10] of fuzzy quantification, that we’ll use here for simplicity, such that $\sigma - count(C) = \sum_{x \in C} \mu_c(x)$ and $\sigma - count(C/D) = \frac{\sigma - count(C \cap D)}{\sigma - count(D)}$. Details of the application of quantifiers to adaptive hypermedia can be found in [19].

$$Q \ c \approx \mu_Q(\sigma - count(c)) \quad (3)$$

$$Q \ b \ are \ c \approx \mu_Q(\sigma - count(c/b)) \quad (4)$$

One possible way of integrating quantified expressions in languages like $\mathcal{ALC}_{FH}$ is treating the result of the quantification as an special assertion. In the absolute case, that assertion has the general form $\langle Q : D = k \rangle$, where D is the concept expression being quantified and k the (current) compatibility grade with quantifier Q similar to an instance assertion. In the relative case, two similar assertions, $\langle Q' : C \sqcap D = k \rangle$ and $\langle Q'' : D = k \rangle$, where C is the base concept expression, so that rules can compute the relative count from them.

For example, we can formalize the concept involved in quantified rule 1 as $\exists belongs.\{x\}^3$, given that $\langle x : WebSite = 1 \rangle$ and *belongs* is used to determine the pages in a site. Then, for a given definition of quantifier *MANY* — usually a concrete S-shaped function — an “special” assertion like:

³ The syntax $\{x\}$ to denote a concept is an extension to $\mathcal{ALC}$.

$$\langle MANY : \exists belongs.\{x\} = 0.6 \rangle$$

may be produced (automatically or on demand). This case operates on a crisp set, but vague set expressions of arbitrary complexity can be used instead, for example, the assertion matching the antecedent of rule 2 can be represented as $\langle MANY : Long \sqcap LOvercrowded \sqcap \exists belongs.\{x\} = 0.4 \rangle$.

Example Rule 1 *if Web Site X has many pages then “some corrective action”*

Example Rule 2 *if many of the Web pages of Web Site X are long and link overcrowded then “some corrective action”*

A relative quantified expression like “most short pages are dense” may be expressed as the pair of assertions $\langle MOST' : WebPage \sqcap Long \sqcap Dense = 0.4 \rangle$ and $\langle MOST'' : WebPage \sqcap Long = 0.6 \rangle$. Since *WebPage* is a crisp category, it's used as a filter for both expressions.

2.3 Flexible Modes of Conjunction

The conjunction of two concepts in $\mathcal{ALC}_{FH}$ is interpreted as the t-norm *min*. This entails a concrete mode of “pessimistic” computing in which, for example, a user belonging to two concepts as $\langle u : A = 0.4 \rangle$ and $\langle u : B = 0.8 \rangle$ result in $\langle u : A \sqcap B = 0.4 \rangle$. This behavior can be reasonable for many cases, but there exists concrete situations in which they produce inadequate results from the epistemological viewpoint. One of this cases is that of concepts or relations that are measured or estimated from others in an imprecise way. This is the case of the measure of usability of a given Web page (i.e. an hypermedia node) $\langle p : WebPage = 1 \rangle$ that may be provided as a metadata item inside the HM. If we use a simple automated usability analysis approach [13] to obtain estimations of the usability of Web pages from quantitative indicators, a possible rule contributing to such inference may be the following:

Example Rule 3 *if page is homepage and page is readable then “some conclusion”*

Assuming that the estimation $Readable \doteq \neg VeryLong \sqcap \neg Dense$ has been defined, the antecedent of rule 3 requires the evaluation for each given page of the $\mathcal{ALC}_{FH}$ concept expression as $A \equiv \neg VeryLong \sqcap \neg Dense \sqcap HomePage$. Let's see a concrete example. If we have the assertions:

$$\Sigma_a = \{ \langle p : HomePage = 1 \rangle, \langle p : VeryLong = 0.1 \rangle, \langle p : Dense = 0.25 \rangle \}$$

following the interpretation $\mathcal{I}$ described above, the maximum degree of the antecedent $Maxdeg(\Sigma_a, \langle p : A \rangle)$ is 0.75. The problem with this result is that it's not fully consistent with expert judgments. Expert studies point out that more complex aggregation operators are required in computing belonging to concepts that are estimated rather than defined intensionally through properties. As an illustration, the example just sketched was subject to a limited study with four

usability experts that resulted in a set of experimental data comprising average estimations for concrete cases in the form of mappings $x_1 \sqcap x_2 \rightarrow y$, being x_1 and x_2 two concrete degrees ($0 \leq 0.1 \leq 0.1 \cdot 2 \leq 0.1 \cdot 3 \cdots \leq 1$) for “not very-long” and “not dense”, and y the estimated appropriate resulting value. Comparing the data set with standard t-norms and t-conorms (min-max, probabilistic and Lukasiewicz), a situation like the one depicted in Figure 2 results in better adjustment properties. The rationale behind such aggregation operator design is that for lower values of both *VeryLong* and *Dense* (i.e. high values of $\neg\textit{VeryLong}$ and $\neg\textit{Dense}$, delimited around the 0.7 frontier), some compensation takes place, while for higher values for both of them, the default pessimistic approach is more appropriate. The other two quadrants in Figure 2 could be represented by a non-standard function with an intermediate behavior such that $\min(x, y) \leq U(x, y) \leq \max(x, y)$.

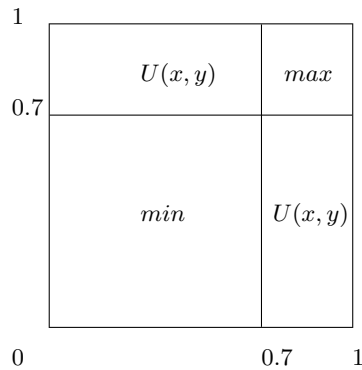


Fig. 2. General scheme for $U(x, y)$

Uninorms [28] are a family of commutative, associative and monotone aggregation operators that allow such forms of compensation, having a neutral element laying anywhere in the unit interval. As such they are good candidates to generalize interpretations of conjunction and disjunction for specific cases of imperfect concept characterization. Of course, the introduction of these kinds of more flexible conjunctions as part of the DL require an analysis of its logical properties that is left to future work. It should be noted also that similar settings may appear in the case of disjunctions.

3 Applications of Vague UI Categories to Adaptive Behaviors

In this section some concrete applications that illustrate the concepts described in the previous section are briefly sketched, namely, preference modeling, infor-

mation filtering problems, and imprecise adaptations on the navigational structure.

3.1 Preference Modeling

Preferences about UI-related descriptions can be modeled from the representations described in the previous section. These preferences may be stated in a general way as assertions in the form $\langle(u, c) : \text{likesGranularity} = 0.8\rangle$ where u is a user of the system $\langle u : \text{User} = 1\rangle$, and c may be a content, node (or any other form of UI element description), which can be expressed as the assertion $\langle u : \text{Content} \sqcup \text{Node} \sqcup \text{UIElement} = 1\rangle$ in a generic way. Once again, storing assertions for the whole cartesian product of users and elements may be unpractical in many situations, specially for large relations stored in databases (see [20]). We have devised an straightforward approach to granulate preferences, consisting on the representation of user preferences as related to prototypical instances representing vague categories. This way, the preference $\langle(u, \text{large} - c_i) : \text{likes} = 0.7\rangle$ states that the given user likes large granularity content items to the given degree.

Then rules like example rule 4 may operate on those descriptions, ultimately combining two membership mappings.

Example Rule 4 *if user likes large content items and current content is c then “some conclusion”*

The computation will proceed by taking the degree of preference to large concepts (0.7 following the above example), and combining it with the degree of compatibility of the content under consideration with the vague granularity category *Large* (say 0.6). Then, some kind of aggregation operator (possibly flexible like the one described in the previous section) must be used to obtain the final preference. This approach may also be combined with other linguistic label aggregation operators like those described in [16], and quantified fuzzy expressions can be used also to estimate preferences as described in the same paper. For example, the rule “if most contents read by the user are short then his preference for fine granularity items is very high” can be used as an heuristic for the implicit collection of preferences.

3.2 Information Filtering

Imprecise preference models like those just described can be used to implement personalized filtering functions acting on UI-related criteria like those described in [16]. The abstract overall form of a filtering function is expressed in 5, where U denotes the set of users, and O denotes the set of the objects (in our case, information items or contents) to be filtered, resulting in a set $\mathcal{R}$ of pairs comprised by an information item and its given relevance degree. Other contextual factors (i.e. the subject of the page being read) may be added to the expression, but they’re content-related, and thus out of our current scope.

$$\mathcal{F} : U \times O \rightarrow \mathcal{R} \text{ where } \mathcal{R} = \{(o, k) | o \in O, k \in [0, 1]\} \quad (5)$$

Preferences can be used as filtering criteria, so that relevances may be obtained by computing $P_{(u_i, o_i)} = \prod_{p \in \text{pref}(u_i, o_i)} p$ for a given object o_j , where $\text{pref}(u_i, o_i)$ is a set containing the values of the grades of the assertions in the form $\langle (u_i, o_j) : \text{likes}X = k_1 \rangle$, or $\langle (u_i, o - X) : \text{likes} = k_2 \rangle$ so that $\mu_X(o_j) > 0$ (or perhaps a threshold-based version $\mu_X(o_j) > w$) if the more compact representation (i.e. the prototypical exemplar version) is used. Once again, the choice for the aggregator of preferences $\prod$ requires further study to be reasonable from the user viewpoint. This form of computing relevances calls for a very specific storage format for preferences, to avoid retrieving all the database of objects. One possible approach may be that of storing the *likes* X predicates as fuzzy relations in compact α -cut format [20] thus obtaining a $O(m+m \cdot \log m)$ complexity, being m the number of relations *like* X from u_i having significant degrees, and provided that the relations are sorted in $m \cdot \log m$ by object to enable the computation of the final degrees in a single pass.

These UI structure-based filtering criteria can be used as a second or complementary filter, combined with content-oriented imprecise filtering expressed in fDLs, like the one sketched in [24].

3.3 Adaptive Linking

Hypermedia links create relationships between two or more addressable items or contents with a concrete semantic or rhetoric intention. If that intention is made explicit, it can be used by the hypermedia system to provide the link with adaptive presentation, hiding or rearranging for specific users [6]. In consequence, ontologies of link types may serve the purpose of codifying common link intentions in a formal way (e.g. as a taxonomy of link types derived from a *Link* concept). But a given node n_i can be connected by link L to a number of destination nodes n_j with varying degrees of strength, as described in [21]. For example, an *DetailLink* may be linked with an strength of 0.8 to a full explanation of a content item, while having a link of strength 0.3 to a brief explanation. This way, the navigation structure itself becomes fuzzy, and imprecise relations can be used to implement diverse adaptations of the UI, e.g. sorting destinations by degree, or filtering only the most relevant ones. In [21], the concept of *fuzzy link* was proposed as a model for such interface elements, expressed in (6), where S and T are the sets of sources and targets (addressable entities), respectively, U is the set of users, and Λ is a set of membership functions.

$$L = (S, T, \Lambda) \text{ and } \mu_{(L,p)} : (S, T, U) \rightarrow [0, 1] \text{ with } \mu_{(L,p)} \in \Lambda \quad (6)$$

The integration of this kind of elements in a $\mathcal{ALCC}_{FH}$ -like language can be carried out according to the following approach. First, links are assertions of link types subsumed by *Link* in HM . Commonly, the intention of a link is fully determined, so that assertions will be in the form $\langle l_1 : \text{AnalogyLink} = 1 \rangle$ for example. A given link will specify as sources or targets a number of nodes of contents also in the HM . These can be expressed as fuzzy or crisp assertions regarding relations, for example: $\langle (l_1, n_1) : \text{source} = 1 \rangle$, $\langle (l_1, n_2) : \text{target} = 0.8 \rangle$,

and $\langle(l_1, n_3) : target = 0.3\rangle$. This entails a model of imprecise hypermedia structure. Personalization can be added in several ways. The simplest approach is the direct assertion of compatibility degrees of links to users, but in common cases, degrees will be computed from established mappings like (7), in which a function of the assertion denoting the (imprecise) *density* of the resource is combined with a function of the assertion regarding knowledge level of the user with the source s to give a degree of compatibility of the link with the user (and where $s \in S, t \in T, u \in U$).

$$\mu_{(L, detail)}(s, t, u) = f(\langle t : Dense = x_1 \rangle) \cdot g(\langle (u, s) : masters = x_2 \rangle) \quad (7)$$

Then, rules using fuzzy assertions may be used to carry out adaptations. Rule 5 describes a typical example.

Example Rule 5 *if the type of the link is `ArgumentLink` and the link (here) is compatible with the user **then** font-size of the link is big and font-family of the link is `Verdana`*

Example rule 5 provides two antecedents. On the one hand, the type of the links for which the rule is applicable is matched against assertions in the form $\langle l : ArgumentLink = 1 \rangle$. On the other hand, the mention of the compatibility grade entails a partial match of the degree of the link. This entails a consideration of the context (i.e. sources, destinations and the current user) of the link usage, by using a predefined function definition in the form $\mu_P : \Sigma_{UM, DM, HM} \rightarrow [0, 1]$ as the one provided in expression 7. In addition, two kind of consequents are provided in rule 5. The second is crisp in nature and will simply change the font type of the anchor (given that an activation threshold is reached), while the first one produce different changes depending on the activation strength. The resulting assertions make reference to personalized links (or better, link appearances), that can be denoted by $U_iLink \triangleq Link \sqcap \exists tailored\{u_i\}$ for user i . This way, the above rule may trigger, for example, the following two assertions: $\langle(l, verdana) : fontFamily = 1\rangle$ and $\langle l : LargeFontSized = 0.7\rangle$ given that $l \in U_iLink$ is the (tailored) link under consideration. The second assertion will entail a (reverse) automated assertion of the actual size in ‘picas’ obtained from an existing concept definition of *LargeFontSized* similar to those showed in Figure 1.

The just described approach can be used also to model the compatibility of users with sequences or links (paths) in the navigational structure, as a possible implementation of graph-based Web structure mining results [15].

4 Conclusions and Future Work

Providing handling mechanisms for vagueness in Semantic Web UI descriptions enables approximate adaptive behaviors that can’t be properly addressed with the standard crisp semantics of DLs. This fact and the imperfection inherent to common Web user profiling techniques [23] point out the necessity of treating

fuzziness as a first-class citizen in personalized Web applications. In consequence, further research should address the integration of fuzzy DLs in the common architecture of adaptive systems. The manifestations of vagueness described in this paper have been implemented in diverse scenarios, and our current work is on integrating them under a common fDL framework.

References

1. Antoniou, G.: A Nonmonotonic Rule System using Ontologies. Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, CEUR Workshop Proceedings, 60 (2002)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Theory, Implementation and Applications. Cambridge (2003)
3. Baader, F., Horrocks, I., Sattler, U.: Description Logics as Ontology Languages for the Semantic Web. In: Hutter, D., Stephan, W. (eds.): Festschrift in honor of Jörg Siekmann. Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin Heidelberg New York (2003)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, **284**(5) (2001) 34–43
5. Bilgiç, T. and Türksen, T.: Measurement of Membership Functions: Theoretical and Empirical Work. In: D. Dubois and H. Prade (eds.) Handbook of Fuzzy Sets and Systems Vol. 1, Chapter 3, Fundamentals of Fuzzy Sets, Kluwer (1999):195–232
6. Brusilovsky, P. (2001) Adaptive hypermedia. User Modeling and User Adapted Interaction, Ten Year Anniversary Issue (Alfred Kobsa, ed.) **11**(1/2), 87–110
7. Brusilovsky, P. and Maybury, M. T.: From adaptive hypermedia to adaptive Web. Communications of the ACM 45 (5), (2002) 31–33
8. Dodero, J.M., Sicilia, M.A., García, E.: A Fuzzy Aggregation-Based Reputation Model for e-Learning Exploitation of Public Domain Resources. In: Proceedings of the Fourth International ICSC Symposia on Soft-Computing And Intelligent Systems For Industry ICSC Naiso Academia Press, Paisley, Scotland (2001)
9. García, E., Sicilia, M.A., Gutiérrez, J.A.: On the Vague Modelling of Web Page Characteristics Regarding Usability. In: Menasalvas, E., Segovia, J., Szczepaniak, P. (eds.): First International Atlantic Web Intelligence Conference. Lecture Notes in Computer Science, Vol. 2663. Springer-Verlag, Berlin Heidelberg New York (2003) 199–207
10. Glöckner, I. and Knoll, A.: A Framework for Evaluating Fusion Operators Based on the Theory of Generalized Quantifiers. In: Proceedings of the 1999 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '99), Taipei, Taiwan (1999)
11. Guarino, N.: Formal ontology, conceptual analysis and knowledge representation. Int. Journal of Human-Computer Studies, **43**(5/6) (1995) 625–640
12. Hölldobler, S., Dinh Khang, T., Störr, H.P.: A fuzzy description logic with hedges as concept modifiers. In: Hoang Phuong, N., Nguyen, H.T., Cat Ho, N., Santiprabhob, P. (eds.): Proceedings of InTech/VJFuzzy'2002, Hanoi, Vietnam, (2002) 25–34
13. Ivory, M.Y. and Hearst, M.A.: The State of the Art in Automated Usability Evaluation of User Interfaces. ACM Computing Surveys, **33**(4) (2001):1–47

14. Ivory, M.Y. and Hearst, M.A.: Improving Web Site Design. *IEEE Internet Computing*, Special Issue on Usability and the World Wide Web 6(2), March/April (2002): 56–63
15. Kosala, R., Blockeel, H.: Web mining research: A survey. In: *SIGKDD Explorations — Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining* 2(1) (2000) 1–15
16. López, L., Sicilia, M.A., García, E.: Personalization of Web Interface Structural Elements: A Learning-Scenario Case Study. In: *International Symposia of Computer Science*. Aguascalientes, Mexico (2001) 579–588
17. McCarthy, J.: Epistemological problems of artificial intelligence. In: *Proceedings Int. Joint Conference on Artificial Intelligence* (1997) 1038–1044
18. Rosch, E. Principles of Categorization. In: Rosch, E., Lloyd, B. (eds.), *Cognition and Categorization*, Lawrence Erlbaum, Hillsdale, NJ (1978) 27–48
19. Sicilia, M. A., Díaz, P., Aedo, I., García, E.: Fuzzy Linguistic Summaries in Adaptive Hipermedia Systems. In: *Adaptive Hipermedia and Adaptive Web Systems. Lecture Notes in Computer Science*, Vol. 2347, Springer-Verlag, Berlin Heidelberg New York (2002) 317–327
20. Sicilia, M.A., Gutiérrez, J.A., García, E.: Designing Fuzzy Relations in Orthogonal Persistence Object-Oriented Database Engines. *Advances in Artificial Intelligence — IBERAMIA 2002, Lecture Notes in Computer Science* 2527 Springer (2002) 243–253
21. Sicilia, M. A., García, E., Díaz, P. Aedo, I.: Learning Links: Reusable Assets with Support for Vagueness and Ontology-based Typing. In: *International Workshop on Concepts and Ontologies in Web-based Educational Systems*. Aroyo, L., Dicheva, D. (eds.): *Technical Report 02-15*, Technical University of Eindhoven (2002) 37–42
22. Sicilia, M.A., García, E., Díaz, P., Aedo, I.: Fuzziness in adaptive hypermedia models. In: *Proceedings of the North American Fuzzy Information Processing Society Conference* (2002) 268–273
23. Sicilia, M.A.: Observing Web Users: Conjecturing and Refutation on Partial Evidence In: *Proceedings of the North American Fuzzy Information Processing Society Conference* (2003)
24. Straccia, U.: A Framework for the Retrieval of Multimedia Objects Based on Four-Valued Fuzzy Description Logics. In: Crestani, F., Pasi, G. (eds.): *Soft Computing in Information Retrieval: Techniques and Applications*. Physica Verlag (Springer Verlag), Vol. 50. Heidelberg, Germany, (2000) 332–357
25. Straccia, U.: Reasoning within fuzzy description logics. *J. Artificial Intelligence Research*, 14 (2001) 137–166.
26. Tresp, C.B., Molitor, R.: A Description Logic for Vague Knowledge. In: *Proceedings of the 13th biennial European Conference on Artificial Intelligence (ECAI'98)*, J. Wiley and Sons, Brighton, UK (1998) 361–365
27. Wu, H., De Kort, E., De Bra, P.: Design Issues for General-Purpose Adaptive Hypermedia Systems. In: *Proceedings of the ACM Conference on Hypertext and Hypermedia* (2001) 141–150
28. Yager, R. R., and Rybalov, A.: Uninorm Aggregation Operators. *Fuzzy Sets and Systems* 80 (1996) 111–120
29. Zadeh, L.A.: A Computational Approach to Fuzzy Quantifiers in Natural Language. *Computing and Mathematics with Applications*, 9(1) (1983) 149–184

Two Layer Web User Interface Analysis Framework Using SNN and iFIN

William Gardner¹, Elizabeth Chang², and Tharam S. Dillon¹

¹Faculty of Information Technology, University of Technology, Sydney, Australia
{wgardner, tharam}@it.uts.edu.au

²School of Information Systems, Curtin University of Technology, Perth, Australia
change@cbs.curtin.edu.au

Abstract. The unique characteristics of Web architecture have given a great degree of flexibility to the design of the Web User Interface (WUI), compared to the user interface design of traditional software applications. This in turn has had some undesirable consequences. The process of WUI design involves both logical and perceptual aspects. Without careful analysis of the UI requirements, usability problems can result in the implemented application. Like the development of any software application, a systematic approach is required to support the design process of WUI; hence the application can support the users to complete their tasks efficiently and effectively. In this paper, we propose an analysis modelling method for requirement engineering of such a Web interface for Web applications. The model can be used for capturing the logical view of the user-system interaction at a high level of abstraction. It is at a sufficient level of abstraction to keep it independent of the particular implementation tool or language.

1 Introduction

In the early days when Internet was beginning commercialised, the idea of Web computing meant putting together a collection of web pages in HTML coded text to form a Web site [CZF99]. The majority of the content contained primarily static information (e.g. company news, product/services description and contact details, etc). The contribution of evolving new Web technologies has transformed the original idea of the Internet, into one that employs it as a platform for distributed applications [BGP00]. Organisations have quickly realised the value and benefit of deploying their applications on a Web-based architecture.

Over the years, several techniques have been introduced in the literature for the modelling and design of Web-based systems. There is a heavy concentration in the earlier methods to be: (1) hypertext oriented [GPS93, ISB95, SRB96] or (2) data centric or data driven [Gre97]. These models were found to not paying sufficient attention to users, who are central in Web systems. These systems, hypertext, or data centred approaches need to be contrasted with the User-centred approach. Two groups of researchers [TL98, CD99] have adopted this user centred or audience centred approach. In recent time, more groups have attempted to take this approach, such as [CGP00]. However the models are frequently at too low a level, and are useful only at

the design stage more than the analysis stage. Few groups have tried to tackle the issue of *dynamics* of the Web site, and in particular Web navigation/user interaction, which remains a major problem in the development of Web applications. The problem with Web navigation has led to big corporations such as Wal-Mart¹, Amazon² to redesign their Web sites to achieve more user-friendly and effective Web navigation. A group that has tried addressing this issue of Web navigation and introduced some important ideas, such as *navigational context*, in addition to some proposed extensions to UML is [PAF01]. The work we present in this paper complements their method in that it is at a higher level of abstraction, concentrates on the analysis model rather than design, and provides a descriptive model with precise semantics utilising extension of Activity diagram, and FINs (a form of High Level Petri Net) [CD98].

In this paper, we propose an analysis model for WUI modelling called the Web User Interaction Analysis Model (WUiAM). The goal of WuIAM is to support the process of the requirement engineering stage for the development of a Web application, by allowing the user-system interaction to be captured and hence analysed.

The organisation of this paper is as follow. Section 2 introduce the WuIAM, this include a brief discussion of conceptual design of Web user interface in Section 2.1, the detail of Site Navigation Net (SNN) and In-context Flow of Interaction Net (iFIN) are given in Section 2.2 and 2.3 respectively. Section 3 shows an example analysis of a Web system using WuIAM. Finally, Section 4 concludes the paper and discusses some future work.

2 Web User Interaction Analysis Model

In [CK96], the term interaction requirement is defined as “*the sequence of interactions between the user and a system*”. We propose here a Web User Interaction Analysis Model (WUiAM), which is a modelling method for representing the possible user-system interaction requirements. This is a systematic approach that allows the specification of an analysis model based on a task/activity oriented approach. The information captured in WuIAM should be isolated from any specific visual or graphical design concerns; it gives a logical view of the WUI that is under consideration. It does not mean to replace some of the currently available conceptual and design modelling methods for Web systems, but as an added set of models targeting the area of user interface of a Web system, which can be integrated and complemented with other domain modelling methods, hence providing a comprehensive system development process. We note here that in addition to *domain analysis*, which leads to a *domain conceptual model*, we are here proposing a method of *user interface analysis* leading to a *logical user interface model*.

An example taken from the development of an e-Business Web application for a large multinational 3rd party warehouse and logistics services provider in the Asia Pacific region will be used in this paper which help to illustrate and explain the idea of WuIAM. The project required the development of an online warehouse

¹ <http://www.walmart.com>

² <http://www.amazon.com>

management system (eWMS). It allows warehouse customers to access their stock information as well as placing different types of operational orders, such as to make a booking for warehouse space, order goods to be delivered from the warehouse and transfer ownership of the goods in the case where both the seller and buyer of the transaction use the same warehouse company. Fig. 1 below shows a simplify version of the use case diagram created during the initial requirement analysis.

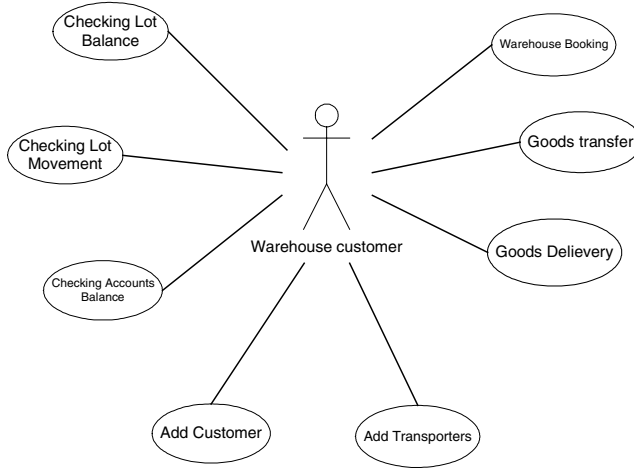


Fig. 1. Use case diagram of e-Warehouse Management System

2.1 Conceptual Design of Web User Interface

The idea of separating the presentation layer (user interface) and the domain layer (business logic) is well established in the software engineering field. An Abstract User Interface Object (AUI) [CD94, CD98] represents a user view on a collection of meaningful information/control objects. This idea of AUI was further extended to characterize Abstract Web User Interface Object (AWUI) in [CD98, CD99]. An AWUI can be differentiated into (i) Web Information Objects and (ii) Web Command Objects. As explained in [CD98], perspectives of domain objects are used to derive these AWUI. This approach provides an analysis model (i) with seamless integration with the domain and server based application or database objects and (ii) that is independent of the implementation tool or platform so allowing its implementation to be replaced. This gives a *layered approach* to Web interface analysis and design.

The conceptual design of user interface for a system would include defining the i) logical flow of user interaction (task & response), ii) the availability of user interface command objects (functional design) and iii) the availability of user interface information objects (information design). Issues of ii) and iii) are of particularly importance to the quality of a Web application [GLS03], and i) will greatly affect usability of the resulting application as perceived by its end user.

2.2 Site Navigation Net

WUiAM contains two layers, namely Site Navigation Net (SNN) and In-context Flow of Interaction (iFIN). The semantic of SNN is heavily based on Activity diagram. However, in order to capture the overall behaviour of a WUI, a number of new constructs are added and modifications are made to the implementation of SNN. The main construct of the SNN is composed of *context*, *sequential transition* and *non-sequential transition*. We will explain each of these components in detail in the following sections.

2.2.1 Context

The concepts of *navigational context* (we will use the term “context” here onward) have been used in a number of existing Web design methods that are discussed in Section 1. A clear and precise discussion is given in [PAF01]. It is a powerful concept for identifying the user view on a subset of information/objects from the conceptual level. This is a useful notion when analysing and designing a Web user interface, as each *context* represents the user perspective of a collection of logically related information and/or objects. In the WUiAM approach, this user perspective produces mapping to what we call an Abstract Web User Interface Object [CD98].

The notion of a *navigational context* is considered as the perspective over a sub-group of objects at the domain level, and user navigation is captured within a context by means of diagrams or schemas [SR98, PAF01]. However, we felt that this notion of context is some what restricted, as it does not visualise the overall user perspective of the complete system. In the WUiAM approach, we define a context as;

Definition *A context is a perspective over a collection of logically related information/control objects, such that a context corresponds to an AWUI, user interaction that occurs within the context and it will result in a change of business state in the system. User action would cause the transition that results in the activation and deactivation of the current context. A context is a generalization of the notion of Activity in Activity Diagram. Features that characterise a context are pre-condition, post-condition, data input, and data output where;*

- *Pre-conditions are conditions that need to be met before a context can become activated, this helps the developer to see the kind of access control that will be required for each context.*
- *Post-conditions are the conditions that must be met before control will be passed onto the next context, these conditions will guard the flow of sequential transition. However, for non-sequential transition these conditions will not need to be met.*
- *Data Input is data that must be passed into the current context from the previous sequential context, if any value is present for this object, it means that a jump link entry to this context will also need to meet this requirement.*
- *Data Output is data that will traverse into the next sequential contexts, again for the situation of non-sequential transition, this condition will not need to be satisfied.*

A *context* is depicted as a lozenge shape with other rectangle box as above and below the lozenge shape (Fig. 2). Essentially, there are two types of user interaction behaviours that can be defined by a *context*:

1. User to System - A major task/activity (e.g. Fill in warehouse booking details) that a user is required to complete in order to achieve a particular goal (e.g. Make a warehouse booking).
2. System to User - A response by the system to the user as a result of some system process that is triggered by user actions.

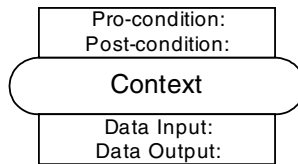


Fig. 2. The representation of a *Context*

Within the lozenge shape is the name of the *context*, this in turn indicates the task/activity that the user performs (e.g. enter booking details). Beside the name of the activity, the control conditions (Pre-condition and Post-condition) are defined in the rectangle box above the lozenge shape and the data conditions (Data-in and Data-out) are defined in the rectangle box below.

There would be a number of minor tasks/activities that need or can be carried out by the user whilst they are within a *context*. The details on how to model the internal flow of interaction for a *context* will be addressed in the section for iFIN (Section 2.3).

2.2.2 Transition

The difficulty in modelling a Web application UI compared to a traditional software system's UI is due to the unstructured nature of the hypermedia space, and the possibility of "switching context" (freedom of movement from one context to another in a non-sequential manner) or what we call 'jump link'. However, the user movement on a web application should be controlled at some degree. Because user behaviour on a web application will not only affect the state of the user interface objects (presentation of information), but also the state of the business process of the system [SR02]. For example, if the user wants to get a quote for an insurance package, which would require 5 major steps (5 *contexts*), the system must keep track of the data the user has entered, and the current state of the process. Any web application should not rely on the navigation feature available on the user's web browser (e.g. In MS Internet Explorer, the 'back' and 'forward' buttons.) as a mean for navigating around the application. Such functionality should always be provided by the UI command objects on the actual *context*. There are two main types of user movement between *context*, namely sequential and non-sequential.

There are two types of sequential transition defined in SNN:

1. Unidirectional sequential transitions: A sequential flow of control focus from the current context to the next context is represented as a solid line with a feathered arrow pointing to the next context (Fig. 3).
2. Bidirectional sequential transitions: A double ended arrow represents the possibility of bidirectional transition between contexts (Fig. 4).

It is important to have each transition marked with its name, the name of a transition acts as the guard for the wiring between the internal iFIN and SNN layer, given that there could be more than one transition leaving from the same context.



Fig. 3. Unidirectional sequential transition



Fig. 4. Bidirectional sequential transition

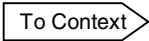


Fig. 5. Jump link (non-sequential transition)

As mentioned earlier, there can also be a non-sequential transition (Jump link) from the current *context*. The representation of a non-sequential transition (Jump link) is shown on Fig. 5 A Jump link would be attached to the context where a possible switch in context could happen (Fig. 6), and the destination context's name is show inside the box. The notation of 'Jump link' can be further defined by a destination tree, that more than one path is contained within this 'jump link'.

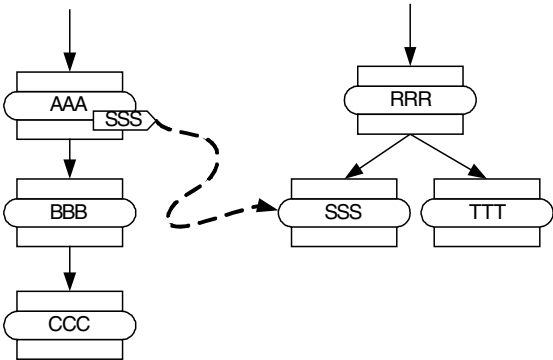


Fig. 6. Jump link representation

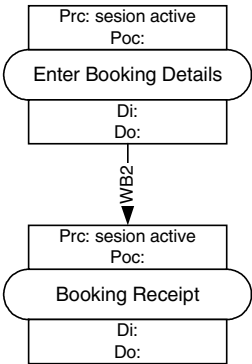


Fig. 7. Contexts defined for the 'Warehouse booking' use case

As an example from the eWMS, the use case 'warehouse booking' will require at least 2 activity contexts, namely 'Enter booking details' and the system will provide customer its 'order receipt'. The representation of this is shown in Fig. 7 which have two sequential contexts.

2.3 In-Context Flow of Interaction Net

In-context Flow of Interaction Net (iFIN) is a modified form of Flow of Interaction Net (FIN) [CD98]. FIN consists of a set of graphical notation proposed for the logical design and modelling of user interface for traditional software applications. It is useful in aiding the design of user interface in such application; however, it needs to be enhanced for modelling of user-system interaction for Web applications.

As discussed in the Section 2.2, SNN only captures the user interaction at the major task level. In order to study the overall navigation pattern, the internal flow of interaction of each context is captured in the iFIN diagram (i.e. each context can be unfolded into a corresponding iFIN diagram Fig. 8). The iFIN captures the minor tasks that need/can be carried out while the user is carrying out an individual major task (context). The initial state of a context is defined by the starting state of its iFIN. It is possible to have multiple sequential and non-sequential exit points in one context, as each end state of iFIN, is wired to the SNN layer by the appropriate guard (transition name) that is shown on the SNN.

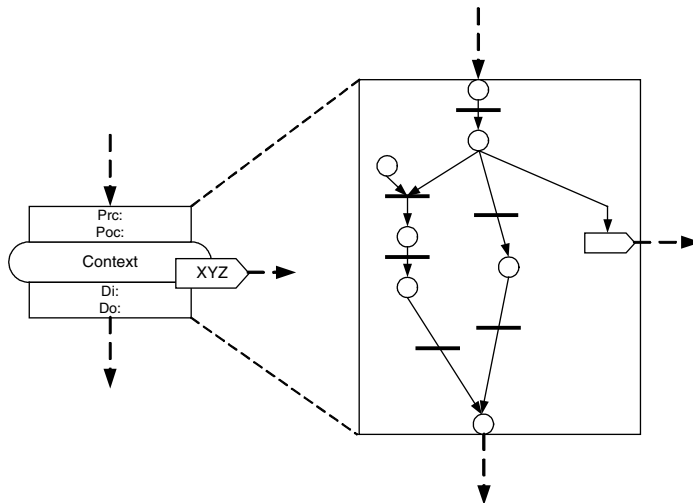


Fig. 8. The idea of iFIN capturing the possible user actions within a context

The current state of the iFIN indicates the current point of control and situation of the activity at the SNN level. For the creation of iFIN, there are three constructs borrowed from the FIN diagram, namely state (Fig. 9a), user action (Fig. 9b), and token (Fig. 9c). In FIN, a state is a representation of the state of a WUI command object, a state before the user action denotes the pre-condition, and the one after denotes the post-condition. A token appears in the state shown, indicate that the current command object is activated, and when all places before a user action are activated, the user action can be fired, and hence cause the transition to occur. For the construction of iFIN, we have added two new notations, they are start/end state (Fig. 9d) and “Jump link” (Fig. 9e) as defined in SNN. These added constructs are required for the complete wiring between the iFIN layer and the SNN layer.

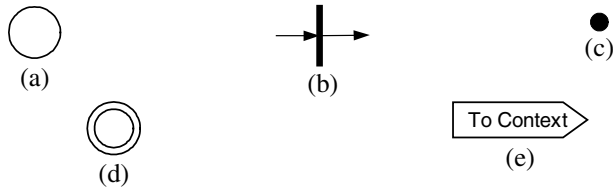


Fig. 9. (a) State (b) User/system action (c) Token (d) Start/End state (e) Jump link

After the initial *context* of ‘Enter booking detail’ is defined, it is clear that this major activity would involve a number of sub tasks. These include ‘Selecting the warehouse’ and ‘Enter the order detail’ (goods arrival date, time, quantity, etc). All these in-context level user interactions are shown in the booking details iFin in Fig. 10.

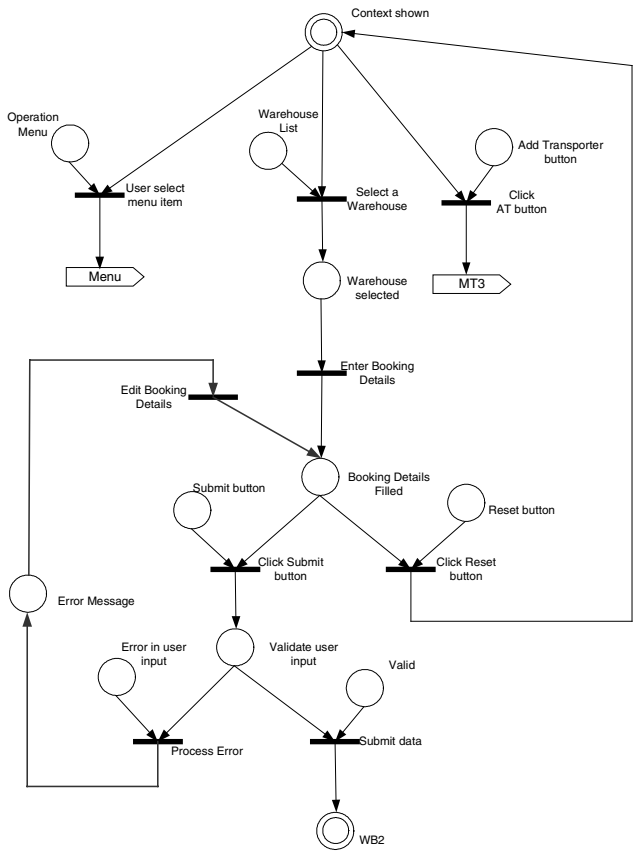


Fig. 10. iFin for ‘Warehouse booking’

The Information that is captured by iFIN also includes the required level of user input validation as it forms an important part of the overall user interface properties, because that would provide details on the necessary system-user response with in the context.

3 Example Problems

The purpose of developing the WUiAM is to provide a means of representing the dynamic aspects of the WUI. It aids in ensuring the design of the actual WUI is free from the problem of 3I's [CD98], that is the WUI would not have problems of illogical (flow of interaction), incorrect (placement of information/command objects), and incomplete (functionality). In the next two diagrams, a complete set of SNNs for the eWMS system example used in this paper are shown. This diagram covers the SNN path of 3 use cases, namely 'Warehouse booking', 'Transfer order', and 'Delivery order'. Fig. 11 shows the initialization of the SNN for these processes, a number of problems were discovered from this first version of the SNN for the system. These are as follows:

- 1. Missing system-to-user response for confirmation of user input.
- 2. Each end task has resulted in a deadend. User would be left with nowhere to go or have to use the back button on the browser.

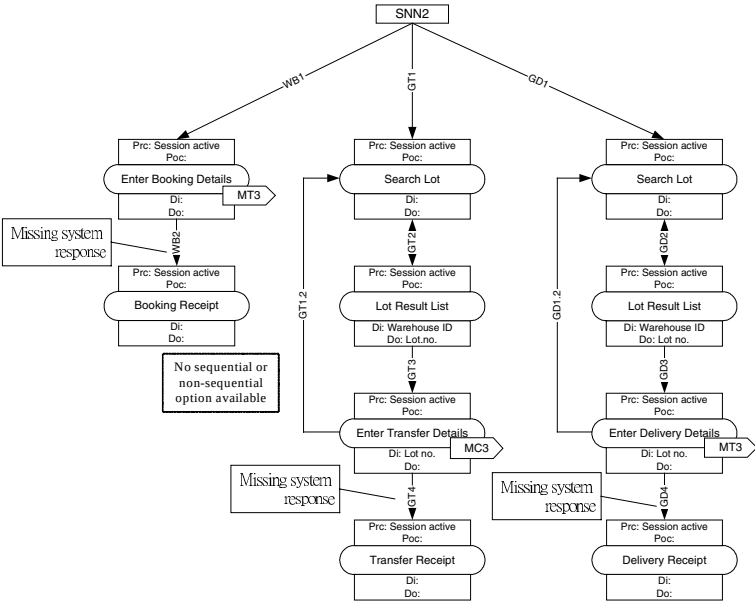


Fig. 11. SNN of 3 use case – warehouse booking, goods transfer, goods delivery

Further studies of the iFIN's of the context 'Warehouse booking' (Fig. 10) and 'Add transporters' also discovered the problem that when user chooses to 'Add transporters' while in the warehouse booking process the user will not be able to return to the previous state and continue the booking order operation. The detection of the problem mentioned above has resulted in the corrected version SNN shown in Fig. 12 and modification to the iFIN of both the 'Warehouse booking' and 'Add transporters'.

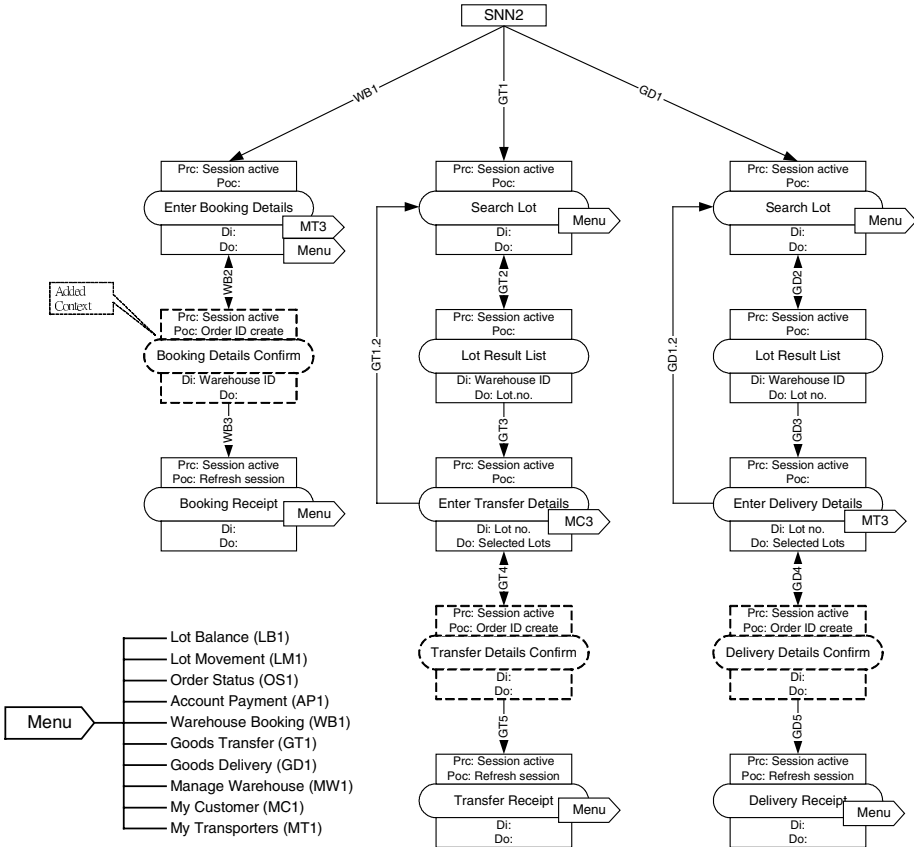


Fig. 12. Final version of SNN for the same use case as in Fig. 11.

4 Future Work and Conclusion

In this paper, we have presented an abstract level analysis model Web User Interaction Analysis Model for the modelling of a Web User Interface. The aim of WUiAM is to provide a simple yet effective method for WUI analyst/designer/developer to capture the user-system interaction for a Web

application at the analysis stage. The construction of the WUiAM is based on a task/goal oriented approach, which allows the discovery of AWUI and hence its integration to a domain model. There is a distinctive difference between the approaches that are used by WUiAM compared with similar models proposed earlier. The two-layered approach with the use of SNN and iFIN provides a means for reducing the complexity and increases the readability of the diagram, and hence assists the identification of navigational problems at an early stage of the development, thereby avoiding such defects from occurring in the Web application. This method have actually been taught and used by the students in the subject Internet Client Side Computing at La Trobe University, Australia. This subject includes undergraduate student in 2nd, 3rd, and 4th year level. The students have produced successful Web interface design by using the WUiAM in aiding the completion of their assignment, which included the design of a University Magazine Web site, or redesign of an Online Book Store Web application. Although, the work of WUiAM is complete, but as mentioned earlier in the paper WUiAM is intended only to be an analysis level model; the next logical step would be the integration of WUiAM with other design level methods. One possible such direction is considered being the idea of generation of user interface code or template. However, the information that is captured within WUiAM alone may not be sufficient for such application. In our future research we will try to address this issue in detail.

References

- [BGP00] L Baresi, F. Garzotto, P Paolini, "From Web Sites to Web applications: New Issues for Conceptual Modeling", ER 2000 Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling, LNCS 1921, 89–100, 2000.
- [CD94] E. Chang, T.S. Dillon, "Integration of User Interfaces with Application Software and Databases through the Use of Perspectives", Proceedings of 1st International Conference on Object-Role, ORM-1, 208–217, 1994.
- [CD98] E. Chang, T. Dillon, "The Navigational Aspects of the Logical Design of User Interfaces", Proc. First International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE ISORC '98, 425–430, 1998.
- [CD99] E. Chang, T. Dillon, "Audience Centred Web Based Design", Proc. 1999 IEEE International Conference on Systems, Man, and Cybernetics, IEEE SMC '99 Conference Proceedings, Volume: 2, 99–104, 1999.
- [CGP00] C. Cachero, J. Gomez, O. Pastor, "Object-Oriented Conceptual Modelling of Web Application Interfaces: the OO-HMethod presentation abstract model", Proceedings of 1st International Conference Electronic Commerce and Web Technologies, EC-Web 2000, LNCS 1875, 206–215, 2000.
- [CK96] B. Chandrasekaran, H. Kaindl, "Representing Functional Requirements and User-System Interactions", 13th National Conference on Artificial Intelligence, Workshop on Modeling and Reasoning about Function, AAAI '96, 1996.
- [CZF99] J. Chen, W. Zhao, X. Fan, "A Content-Centric Modeling Approach for Web-based Application", Proc. 2nd Asia Pacific Web Conference, Hong Kong, 3–9, 1999.
- [GLS03] A. Gu, D. Lowe, B. Henderson-Sellers, "Linking Modelling Capabilities and Abstraction Levels: The Keys to Web System architectural Integrity", The 9th Australian World Wide Web Conference, AusWeb03, 2003.

- [GPS93] F. Garzotto, P. Paolini, D. Schwabe, "HDM - A Model-based Approach to Hypertext Application Design", *ACM Transactions on Information Systems*, 11(1), 1–26, 1993.
- [Gre97] F. Greenspun, "Database Backed Web sites: The Thinking Person's Guide to Web", SD Press, 1997.
- [ISB95] T. Isakowitz, E. Stohr, P. Balasubramanian, "RMM: A methodology for structured hypermedia design", *Communication of the ACM*, 8(38), 34–44, 1995.
- [PAF01] O. Pastor, S. Abrahao, J. Fons, "An Object-Oriented Approach to Automate Web Application Development", In *Proc. 2nd International Conference, EC-Web 2001*, LNCS 2115, 16–28., 2001.
- [SR02] H. A. Schmid, G. Rossi, "Designing Business Process in E-commerce Applictaion", *Proceedings of 3rd International Conference on E-Commerce and Web Technologies , EC-Web 2002, France, LNCS2455*, 2002.
- [SRB96] D. Schwabe, G. Rossi, S. Barbosa, "Systematic Hypermedia application design with OOHDM", *Proceedings of the ACM International Conference on Hypertext*, 116–128, 1996.
- [TL98] O.M.F. De Troyer, C.J. Leune, "WSDM: a User Centered Design Method for Web Sites", *7th International World Wide Web Conference, Australia*, 1998.

Usability Metrics for E-learning

Siu Keung (Brian) Wong, Thao Thi Nguyen, Elizabeth Chang, and Nimal Jayaratna

School of Information Systems,
Curtin University of Technology, Perth 6845
skwong2@optusnet.com.au,
{change, nguyent, nimal}@cbs.curtin.edu.au

Abstract. E-learning is a fast-growing area thanks to the potential for increased reach and decreased cost of e-learning programs. Usability is of vital importance for e-learning due to the remote nature of the e-learning activities. In order to evaluate usability of e-learning, we apply the following usability factors: 1) E-learning System Feedback, 2) Consistency, 3) Error Prevention, 4) Performance/Efficiency, 5) User like/dislike, 6) Error Recovery, 7) Cognitive load 8) Internationalization, 9) Privacy, 10) On-Line Help [14, 15]. We noticed that we needed a way of representing each of these factors separately and finally to combine them into a single figure of merit expressing overall e-learning usability. Note each of the above dimensions is a relatively imprecise concept and cannot be simply represented numerically. Hence, we decided to use Fuzzy system theory as the basis for representing them and the Fuzzy-reasoning paradigm using the Takagi-Sugeno (T-S) model [15] for combining them into a single e-learning usability figure of merit. This approach has 3 components: a) identification of the usability issues in e-learning, b) a testing aspect for the Fuzzy based usability approach, c) combining the results of these tests and turning them into a usability measure and the validation of the Fuzzy model. This paper discusses the use of fuzzy systems to model each of these usability factors and to reflect how each affects the overall value of e-learning usability.

1 Introduction

E-learning could be defined as learning activities involving computer networks [22]. The networks could be intranets, extranets, or the Internet. E-learning has been made possible by the advancement of the Internet and multimedia technologies. However, it is the increasing number of learners, coupled with the limitation of resources available for education and training that has contributed to the growth of e-learning. The market for e-learning is forecasted to grow from \$14.9 billion in 1998 to an estimated \$33.7 billion in 2004 [18].

The increasing popularity of e-learning could be credited to a number of its important benefits. First, e-learning offers the potential for substantial cost savings in the provision of learning programs. Once developed, learning components such as learning materials and learning objects could be reused. Second, e-learning allows the learners the flexibility of learning anytime, anywhere instead of being restricted to a certain time and place. Finally, organizations can use e-learning to provide “just-in-

time” learning facilities where employees can access the right knowledge when they need it [17, 18].

Due to the remote nature of e-learning, usability plays a vital role in the success of an e-learning program. An e-learning system with poor usability hinders e-learning efforts as the learners would spend more time learning how to use the software rather than learning the contents. Yet, it is a concern raised by a growing number of experts in the current literature that usability and its evaluation have not yet been given due attention in the development of e-learning systems [19, 20, 21]. There is a real need for a usability evaluation method that is practical and specific to e-learning.

Usability of a system “is measured by how easily and effectively it can be used by a specific set of users, given particular kind of support, to carry out a defined set of tasks, in a defined set of environments” [16]. The current approaches to evaluating the usability of a system could be categorised into: (a) Empirical testing [5]; (b) Inspection [6]; (c) Comparative usability measures [1]; (d) Formal complex based measures [9]. Empirical testing entails conducting usability evaluation in a fairly controlled situation to identify the problems that the user have with the system. Inspection involves examining the design of various screens to identify problems that might occur in the user interfaces. Comparative usability measures consists of usability metrics that are designed to evaluate aspects of the interface such as the ability of the user to perform the tasks that s/he is required to do efficiently, and acceptance by the user of the user interface. Finally, formal complexity based measures on the other hand use a variety of criteria that rely on some formal or mathematical analysis of the user interface.

In this paper we discuss the development of a comparative usability measurement approach for e-learning systems that has three components, namely, (a) a heuristic evaluation, (b) a testing aspect, (c) a usability measure that consists of the results of the tests. The framework we employ for developing the usability measure is Fuzzy Systems.

2 Problem Definition and Choice of Methodology

Evaluation of the usability of e-learning could be used to isolate learning problems with particular e-learning packages and propose methods for correcting these and/or to obtain an overall figure of merit of the usability of the system to allow comparisons between different-learning systems. The focus of the usability measurement approach discussed in this paper is on obtaining the figure of merit for all aspects of the e-learning system and for each different aspect of the e-learning system.

Usability of an e-learning system is a difficult concept to quantify as it involves fuzzy concepts and multiple dimensions and factors that could be subjective. When examining usability, Dillon and Maquire [1] isolated four major aspects (a) user performance (b) user attitude to the system (c) mental effort required and (d) formal analysis of the system to obtain operational complexity. Each one of these dimensions themselves can be further decomposed. User performance can be measured by time taken to complete the task; it could also be measured by user efficiency or relative efficiency which is the ratio of user efficiency divided by expert efficiency. Similarly, measurement of user's preference and attitude can itself be decomposed as described in the Metrics for Usability Standards in Computing (MUSIC) system [1, 2]. So when

one examines each of these, one notices that there are several factors which go into the mix for characterising usability. In this paper, the basic problem we intend to address is to isolate the most important of these factors and work out a means of characterising each of them individually. And then to develop a technique which seeks to combine them in such way that we obtain a composite overall figure of merit for the total e-learning system, as well as a figure of merit of the first three dimensions referred to above, namely the actual user performance, user preference and cognitive load. The difficulty in combining these factors lies in the fact that each one of these factors on their own is an imprecisely defined concept and has fuzzy aspects. Furthermore the notion of usability itself cannot be readily quantified into a numerical value, which can be immediately understood as a figure of merit. The notion of usability itself has a number of fuzzy aspects. For these reasons we have chosen to examine fuzzy systems theory as a mechanism for characterising each of these individual factors and dimensions and also for characterising usability itself. In the rest of this paper, we will design a measure of usability and measures of each of the dimensions of the usability based on fuzzy system theory. This fuzzy system based usability measure can be utilised in the prototyping and implementation stages to characterise and improve the user interface.

3 Fuzzy System Based Usability Metric for an E-learning System

In order to characterise usability of an e-learning system, we apply these factors [14, 15]: 1) E-learning System Feedback, 2) Consistency, 3) Error Prevention, 4) Performance/Efficiency, 5) User like/dislike, 6) Error Recovery, 7) Cognitive load 8) Internationalization, 9) Privacy, 10) On-Line Help. We will now discuss each of the factors to determine how they can be measured.

1. System Feedback

The system should always provide users with the appropriate feedback so that they can be informed about what is going on in the system at any time.

System feedback is characterised by a number of aspects including:

- Error localisation
- If an action is not permitted, does the system say why?
- Does the system give prompts as to what to do next?
- Does the system let one know where one is?
- Does the system tell one why one can not do something?

Inadequate system feedback has several components and these include:

- a. No. of times dialogue/feedback is missing
- b. No. of times dialogue/feedback unnecessary
- c. No. of times system feedback confuses the user (information, tasks, path, message, help)
- d. No. of messages that are irrelevant
- e. No. of actions taken which lead to repeat of the same system feedback message
- f. No. of times the user makes the wrong choice of action based on system feedback
- g. No. of times user grumbles about the feedback messages

Each time anyone of these components is not satisfied, we can say that on this occasion system feedback was inadequate. Thus we will count this as contributing one unit to a measure called inadequate system feedback. System feedback itself is usually characterised using linguistic terms such as Good, Average, Poor.

The Fuzzy Term Set therefore is {Good, Average, Poor}. If we employ triangular shapes for the membership functions, the Fuzzy membership feedback will look like Fig. 1a.

It is useful to discuss each of these aspects of inadequate system feedback a-g in more detail in order to understand exactly what they mean and how they are gathered. We therefore discuss them in turn here.

During the test, the tester expects some feedback at various times. If the system does not provide feedback when the tester expects it, we take the situation as contributing one unit to the overall inadequate system feedback. An example of this would be when the user wishes to carry out a print. This takes a finite amount of time for the system to spool the file followed by the print. If the system does not provide the system with any feedback, either with a clock or an hour glass indicating that it is in fact processing something or with a message such as print is currently being spooled, the user could believe that the system has hung up.

Unnecessary system feedback can distract the user and clutter the screen. Therefore a count is done on the number of times a feedback message is given by the system when the user finds that it is unnecessary. Again each occurrence of this contributes one unit to the value of overall inadequate system feedback.

The system feedback could confuse the user either because it is not precise enough or the language it uses is not familiar to the user. Furthermore it could also be obscure to the user because its size and location on the screen is such that it does not easily attract the user's attention. He could therefore be led into carrying out unnecessary actions such as choosing incorrect options because it is unclear what he is to do next.

A message could appear which is not relevant to the particular state that the system is in, or particular set of actions that he needs to carry out. This could distract the user and mislead him into believing that he is in a state other than the one that he is currently in. Each irrelevant message contributes one unit to the overall inadequate system feedback.

If the system provides the same feedback message for different user actions, the user could be misled into thinking that the tasks they are currently doing is the same as the previous task, which resulted in the same feedback message. There could, of course, be differences in the tasks, and the feedback messages concerned should reflect these differences. Each time a different user action gives the same feedback message, it is taken to contribute one unit to the overall inadequate system feedback.

If the user, after reading the system feedback, takes an action which is erroneous or wrong, it implies that the system feedback message is confusing the user or it is unclear as to the course of action he should pursue. Each occurrence of this contributes one unit to the overall inadequate system feedback measure.

This component (g) represents the number of times that the user is annoyed by or dissatisfied with the feedback message. It is measured by asking the user to grumble out aloud on each occasion that he is unhappy about the system feedback message. Each occurrence of this contributes one unit to the overall system feedback.

2. Consistency

The look, feel and behavior of the interface should be consistent throughout the application and also consistent with other applications in the same domain. Most guidelines, such as those referred to earlier in the paper, seek to bring this about. This consistency should be across a variety of issues such as message display methods, colour use, key definition, data entry methods, etc. If the user interface is consistent it reduces the amount of uncertainty that the user faces when using the interface. It also is likely to reduce the number of erroneous interpretations or actions that the user makes. Consistency of the interface has a number of components and these include consistency with the respect to the following:

- a. Message display methods (prompts, warnings, help)
- b. Colour use (entry form, menu and submenu, foreground/background)
- c. Keys definition
- d. Data entry method
- e. Menu, dialogue and window display methods
- f. Menu hierarchy is consistent with the real world
- g. Terminology used is the same as in real life in that domain
- h. Menu options have to be consistent with Menu Title.

Each time anyone of these is not satisfied between screen to screen displays, we will count this as contributing one unit to a measure called inconsistency. The Fuzzy Term Set is {Good, Average, Poor}, and the Fuzzy membership for consistency will look like Fig. 1b.

It is useful to discuss these components of consistency in a little more detail, in order to assist the person collecting information for this factor of overall usability. Consistent message display methods require that any message in each category gives a prompt, a warning or help, has the same format, and is different from messages in other categories. The format should be similar in presentation as well as colour use. This will ensure that if a particular warning comes up, or if a particular help message comes up, the user is immediately aware of the type of message.

The system user interface has to be consistent in the use of colour with the respect to the following: a) work space, b) text, c) menu bar, d) tool bar, e) menu text, f) inactive title bar, f) active/inactive border (for programs where multiple applications can be run simultaneously), g) highlighted text, f) background.

Inconsistent colour use can be irritating to the user as well as distract him and even mislead him as to the significance of something on the screen. Any special or function keys which are used must be defined consistently. For example, the "ESC" key might be used to abort a function. If, on another occasion the user uses some other key, say an exclamation mark, to abort a function, then the user could be confused and would have to learn the specific special keys used in a specific situation, rather than knowing that a particular key is used to perform a similar operation in all situations. As mentioned above, the user interface should be consistent with respect to data entry methods and be consistent with respect to menu dialogue and window display methods. If, for example, a dialogue box is used for data entry, or a pull down menu with selection from a list of files is used, then that approach should be taken throughout the application.

The terminology used should be similar to that which the user is likely to find himself using in the real world. A mistake often made is that terminology used reflects the real world of the system developer rather than the real world of the user.

An example of confusing terminology would be the use of the notion default which to the user means that a variable, if not specified, utilises the default value. However in the banking environment default to a loan adviser would indicate that the loan had not been met. Lastly menu options should be consistent with the menu title. For example, if one has a pull down menu which says "file", it would be inappropriate to put in "date" as a menu option which would allow one to change the date display.

3. Error Prevention

Error prevention is an important goal of the design of the user interface. If the user interface specifically helps the user to avoid making errors, it increases his efficiency. It will also reduce the level of frustration the user is likely to experience with the user interface and therefore bring about greater acceptance of the user interface by the user. There are several aspects which need to be taken into account when measuring error prevention and these include the following:

- a. No. of errors encountered during task.
- b. No. of wrong key strokes/press causing Error Messages
- c. No. of times the same key is pressed without the desired response from system
- d. No. of extra key presses that are unnecessary
- e. No. of times the same error is encountered
- f. No of steps missing compared with real world execution

Each time anyone of these is not satisfied, we count one unit towards error occurrence. The Fuzzy Term Set is {Good, Average, Poor}, and the Fuzzy membership for error prevention will look like Fig. 1c.

Most of the above components which make up the measure of error prevention are self explanatory. It is, however, worth detailing a few of them in order to understand the differences between some of these components. (a) is simply a measure of the number of errors that have occurred during a particular task being conducted by the user when carrying out usability testing. If the number of errors is low, the error prevention of the user interface, at least for that task, is good. If the number of errors is high the error prevention is poor. (b) really relates to the user trying to perform an illegal operation which, instead of carrying out the operation which the user thinks will be carried out, leads to an error message. (c) on the other hand leads the user to believe that a certain key press will produce a certain desired response from the system, i.e. he believes that the key is linked to a certain operation by the user interface, however, that key is not in fact linked to that operation but an alternative key might be. An example of that would be to use the arrow keys to move across to the next desired button in the user interface rather than using the TAB key. (d) relates to occasions when the user believes that a key press is necessary but in fact the key press has no effect. (e) relates to the user in fact generating the same error again and again. (f) relates to the degree of closeness between the steps which the user goes through in the user interface and the carrying out of that similar task in the real world. If there are steps missing, then it is probably more likely that the user would imagine that they are in place and seek to execute them, leading to generate one of the above error components (a) to (e).

4. Performance/Efficiency

Performance or efficiency is a quality of the user interfaces which characterises how effectively or efficiently the user can complete his tasks. Performance and efficiency has a number of components and these are as follows:

- a. No. of goal/tasks not achieved
- b. Time taken for task completion
- c. Unproductive period
- d. % of task not completed.

Each time anyone of these is not satisfied, we count one unit towards inefficiency. The Fuzzy Term Set is {Good, Average, Poor}, and the Fuzzy membership function will look like Fig. 1d.

To elaborate on the components which go into making up the performance or efficiency measure, we note that during usability testing the user is given a number of tasks to complete.

The first component measures the number of goals or tasks that the user is unable to complete in the time given for usability testing. The second component actually measures the time taken to complete the particular task. Tasks seem to vary in duration and complexity and therefore just using the actual time taken may not be a good enough indicator, because it does not contain the notion of the length and complexity of the task. In order to develop the measure of the task completion time, we use the ratio of the time taken by the user divided by the time taken by a known expert user. Therefore:

$$\text{Task completion time} = \frac{\text{Time taken by user to complete task}}{\text{Time taken by expert user}}$$

For every ten per cent increase of the task time taken by the user over the task time taken by the expert user, we will count as one unit towards the number of inefficiencies involved in task completion time. If the time taken by the expert user was 100 minutes and the time taken by the user in the usability test was 130 minutes then we would take the contribution towards efficiency caused by task completion time as 3 units.

Unproductive period represents time spent by the user not actually working on the specific task or not working towards the specified goal involved in the task. It consists of time that the user spends referencing help, thinking about what to do next and solving problems not directly associated with the task at hand. Unproductive period is defined by reference [2] as the formula:

$$\text{unproductive period} = \text{help time} + \text{search time} + \text{snag time}$$

The last component consists of the percentage of tasks the user did not actually complete and it is calculated directly from the first measure of goals not achieved and is given by:

$$\text{Percentage of task not completed} = \frac{\text{Goals not achieved}}{\text{Number of goals expected to be achieved}}$$

5. User Like/Dislike

Unlike the previous measure which measures the manner in which the user interface facilitates user effectiveness or efficiency, the like/dislike factor measures user

preference. This essentially indicates the level of satisfaction which the user feels with the system and the user interface. There are in fact two alternative approaches which can be used to measure this user like/dislike.

Approach 1:

- (a) Count number of times user gives positive comments when testing the system.
- (b) Count number of times user grumbles or gives negative comments.

Approach 2:

Ask user to indicate on a scale of 1 to 10 the likeability of the system.

From approach 1, we add one unit to user likes for (a) or we subtract one unit for (b). In approach 2, we use a scale of 1 to 10 and convert it using the membership function into the fuzzy sets. Fuzzy Term Set therefore is {Good, Average, Poor}, and the Fuzzy membership function will look like Fig. 1e.

6. Error Recovery

Error recovery is that quality of the system of the user interface which allows the user to exit from a situation that the user did not intend to be in. Users frequently choose the wrong option or enter the wrong data and they are likely to find themselves in an error state from which they need to recover. The manner in which the system facilitates this error recovery could reduce the time the user spends recovering from this error state. Error recovery consists of a number of components and these include.

- a. No. of times the user has to redo the task
- b. No of time the user did not continue
- c. No of actions taken that do not solve the problem
- d. No of minutes (hours) spent on one error recovery
- e. % of all time spent on error recovery
- f. No of times the user has to reboot/start again

Each time anyone of these is not satisfied, we count one unit towards inadequate error recovery. Fuzzy Term Set therefore in {Good, Average, Poor}, and the Fuzzy membership for error recovery looks like Fig. 1f.

The first component essentially means that the user made sufficient errors in a particular task requiring that he has to redo it completely. The second measures the number of times the user has got himself into a condition whereby he is unable to continue the task any further. This state could be the result of the user carrying out an action which was not intended by the system designer but caused the system to enter an error from which there was not an emergency exit to allow the user to continue the task. The third component measures the number of actions which were unsuccessful that the user takes to get out of an error state. For example if a user finds himself in an error state and he needs to press a certain key to get out of it, if the key is an expected key like ESC he will be able to get out of the error straightforwardly and proceed. If, however, another key was chosen which was obscure and the user was unfamiliar with the user interface, he may end up in the error state trying out a number of keys before he finally hits upon the specific key that allowed him to exit. The fourth component represents the amount of time the user is actually in the error recovery state. We measure the time in minutes so we count each minute as contributing a single unit to an inadequate recovery. The fifth component measures the proportion of the total time to complete the required tasks that the user spends trying to recover from errors. The last component indicates the system has got into such an error state that the only option for the user is to reboot the system.

7. Cognitive Load

Cognitive workload is related to the mental effort required by the user to perform tasks using the computer system. If the cognitive workload is high then the user is likely to experience a degree of stress and a degree of pressure in using the particular user interface. If this degree of stress and pressure is unacceptably high, then the user interface needs to be improved in respect to this particular aspect. There are a series of measures to gauge cognitive workload, some of which are objective and others subjective. The objective measures largely measure heart rate and respiration and they require instrumentation of the user and therefore could be intrusive in actually achieving a proper understanding of the users cognitive workload, in the sense that the user could react to the electrode attachment rather than just the user interface. Subjective measures, on the other hand, utilise questionnaires and look at a series of factors to determine the cognitive workload. Two widely used questionnaires are the subjective mental effort questionnaires (SMEQ) which were developed at the University of Kronnigon and Delft University of Technology and the task load index (TLX) which was designed by NASA. The approach explained here could be utilised with either of the above two questionnaires or alternatively with other questionnaires. We will utilise a count on a series of components to help determine the cognitive workload. However the methodology used here could be easily adapted to be used with any of the subjective measures, or indeed the objectives measures of cognitive workload. The components that we will look at to determine the cognitive workload are as follows:

- a. No. of unfamiliar concepts/terminology one has to use
- b. No. of unmatched task executions with the real world
- c. No. of hours/minutes spent on the first glance of user manual
- d. No. of times the manual is used
- e. No. of times the user has to access the help window
- f. No. of times one has to ask for help from help desk personnel
- g. No. of things the user has to remember rather than select on the screen

Each time anyone of these is not satisfied, we count one towards cognitive load. The Fuzzy Term Set therefore in {High, Average, Low}, and the Fuzzy membership for cognitive load will look like Fig. 1g.

If the user comes across unfamiliar concepts or unfamiliar terminology then he is likely to experience a higher cognitive workload than if he is only working with familiar concepts or familiar technology. The same applies if we come across task execution sequences which do not match with the real world. If the user seems to rely very heavily on the user manual, either in terms of spending a lot of time with the user manual initially before he starts to use the system, or if he has to revert to using the user manual frequently or to access the help window frequently, then the user is experiencing a higher cognitive workload than if he did not have to do these things. Also, if he has to keep accessing help desk personnel for assistance, then he is in fact experiencing a higher cognitive workload, in the sense that he is unable to directly understand the user interface. If the user interface is such that the user is frequently faced with a selection of items on the user interface rather than having to remember cryptic commands, then the user has a lower cognitive workload. If, on the other hand, he has to remember things like OPTION C for making a copy, or CONTROL C for making a copy rather than just selecting Copy from the menu, then s/he has a higher cognitive workload than in the second case.

8. Internationalization

One of the differences between an e-learning application and a traditional application is Internationalisation. A strength of e-learning is the ability of providing education to remote students who could be from different countries. In order to be able to do this, the e-learning system needs to be designed to handle different languages and cultures. Thus, a good e-learning system should consider the nationality of the users. Internationalisation has a number of components with the respect to the following:

1. Number of times that the user is required to use a language which is unfamiliar;
2. Number of times that the user encounters a situation where s/he doesn't understand what the e-learning system means due to a nationality problem;
3. Number of times that the user's nationality leads to a dislike of the user interface.

Each time any one of these is not satisfied, we count one towards low internationalisation. The Fuzzy Term Set is: {Good, Average, Poor}, and Fig. 1h shows the fuzzy membership function for Internationalisation.

Point 1 is simply about asking the user to use a particular language to view and/or submit information through the e-learning system. For example, the e-learning system may ask the user to use only English to complete an application form.

Point 2 concerns a situation where the user cannot understand what the e-learning system means because of his/her nationality. For example, if an e-learning system wants to gather some information about sport. The e-learning system may ask "Do you like football?" This question may be confusing to a user from Hong Kong to whom football means "soccer" – a word that has a totally different meaning in Australia. This confusion occurs because of different cultures.

Point 3 is simply about things that the user dislikes about the e-learning system due to his/her culture and/or nationality. For example, people from Hong Kong may not like to see a white and black background as this has funeral connotations.

9. Privacy

Privacy is one of the important factors on Usability Testing. Low security on privacy or force user to let their privacy to be public will drive users don't like your web system at all. In other hand, if you let users have choice to choose whether they allow web system to publish their information or not, then your site will be much popular than the other. Thus, privacy is important while we build up a web system on Internet. Privacy has a number of components with the respect to the following:

- a. No of time that they require your privacy information as compulsory, which is not really gathering as security or functionality purpose.
- b. No of time that the web user interface pops up advertising or non-relative material that is not requires to complete the task.
- c. No of time that the web system sends you junk emails, which the web system did not ask for your permission yet.

Each time anyone of these is not satisfied, we count one towards privacy. The Fuzzy Term Set therefore in {Good, Average, Poor}, and the Fuzzy membership for privacy will look like Fig. 1i.

(a) Although web system used to required user information for security purpose such as surname, date of birth has to be filled before it provide web service. But sometime, web system only attempt collect marketing information by asking user to fill every single field to proceed further. The manner makes users feel annoying. Thus

every time when web system asking some information that are not really relative to security purpose or web functionality need such as password, username etc, we count one toward to low privacy.

(b)Pop up advertising frame that is not relative to the task that the user actually performs without asking users' permission will drive user feel annoying. Thus we count one toward to low privacy every time when the unnecessary frame such as advertising.

(c)One of the problems that users always complain is not always received junk emails, which they don't even want it from web system. Thus we count one toward to low privacy every time when send junk emails without asking users permission.

10. Online Help

Online help is one of the essential functions that systems must provide to users so that they can receive assistance when using the system. Thus, a good e-learning system should have a help section that can be accessed by the user as required. Help has a number of components with the respect to the following:

1. Number of times that help is missing within the e-learning page when the user requires help;
2. Number of times that users cannot find help in the general help sections.

Each time either of these is happens; we count one towards poor help. The Fuzzy Term Set is: {Good, Average, Poor}, and Fig. 1j shows the fuzzy membership function for Help.

A user would always feel annoyed if s/he could not get help straight away from the on-line learning system that s/he is actually using. Thus, if the e-learning system can provide useful help on what the user needs by s/he clicking on the same page of an e-learning package, s/he will feel happy to use the system. Conversely, if s/he cannot find help on that page, even if s/he eventually does find help in the general help section, the user may feel that s/he has had to take too much trouble. A user will complain if s/he cannot find help from their chosen page.

4 Usability Testing

The primary aim of the usability testing for e-Learning system conducted within this project is to generate an extensive set of data, which is to be used to help form the various control rules. The data obtained from the usability testing is to help form the breakpoints of each of the subspaces for each of the usability factors. The results from the test will be used to define the control rules and the breakpoints as accurately as possible, so as to provide a reliable method of determining the usability measure of newly implemented interfaces.

4.1 Fuzzy Rule Base for the Usability Problem

As explained above we intend to use the first order Takagi Sugeno approach for fuzzy inference with a linear function for the right hand side. The inputs on the left hand side of the fuzzy rule will consist of the factors or features that affect usability defined

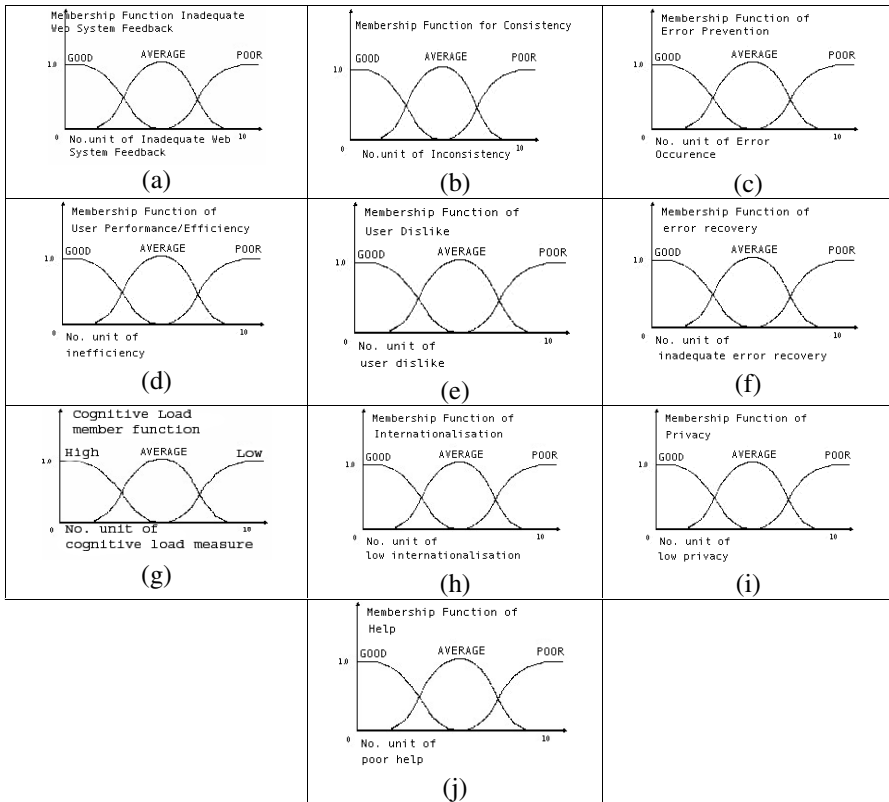


Fig. 1. Member functions

in Section above. These factors or features are WEB SYSTEM FEEDBACK, CONSISTENCY, ERROR PREVENTION, USER PERFORMANCE, USER LIKE/DISLIKE, ERROR RECOVERY, COGNITIVE LOAD, PRIVACY, INTERNATIONALISATION, and HELP.

The input vector x is, therefore, defined to be $x = [\text{SYSTEM FEEDBACK, CONSISTENCY, ERROR PREVENTION, USER PERFORMANCE, USER LIKE/DISLIKE, ERROR RECOVERY, COGNITIVE LOAD, PRIVACY, INTERNATIONALISATION, HELP}]$. We will write this using short form notation as

$$x = [\text{FB, CO, EP, PF, UO, ER, CL, PR, IN, HE}]$$

A typical premise would be of the form "Feedback is good".

The structure of the left hand side of the rule, therefore, is

IF ((WEB SYSTEM FEEDBACK is X_1) AND (CONSISTENCY is X_2) AND (ERROR PREVENTION is X_3) AND (USER PERFORMANCE is X_4) AND (USER LIKES/DISLIKES is X_5) AND (ERROR RECOVERY is X_6) AND (COGNITIVE LOAD is X_7) AND (PRIVACY is X_8) AND (INTERNATIONALISATION is X_9) AND (HELP is X_{10}))

X_i , $i = 1, 2, 3, 4, 5, 6, 8, 9, 10$ denotes in each case the fuzzy sets corresponding to the linguistic terms [Poor, Average, Good].

X_7 denotes the fuzzy sets corresponding to the linguistic terms [Low, Average, High].

The actual form of the fuzzy sets and the associated membership function in each case were defined in Section 2. In short form notation the left hand side of each of the rules would take the form:

IF ((FB is X1) AND (CO is X2) AND (EP is X3) AND (PF is X4) AND
(UD is X5) AND (ER is X6) AND (CL is X7) AND (PR is X8) AND
(IN is X9) AND (HE is X10)).

Since we are using the linear form of the first order Takagi Sugeno inference system the right-hand side for the rule q has the form

Web usability $y_q = a_q0 + a_q1 x1 + \dots + a_q10 x10$, where $x1, \dots, x10$ are the input variables

i.e. in short form notation –usability:

$y_q = a_q0 + a_q1 * FB + a_q2 * CO + a_q3 * EP + a_q4 * PF + a_q5 * UD + a_q6 * ER + a_q7 * CL + a_q8 * PR + a_q9 * IN + a_q10 * HE$

Here $a_q0, a_q1, \dots, a_q10$, are parameters. A typical example of the left hand side in short form notation is

IF ((FB is GOOD) AND (CO is GOOD) AND (SN is GOOD) AND (EP is GOOD) AND
(DT is AVERAGE) AND (PF is AVERAGE) AND (UD is GOOD) AND (ER is GOOD) AND
(CL is AVERAGE) AND (PR is AVERAGE) AND (IN is GOOD) AND (HE is POOR))

The form of the q th rule in the rule base, in short form notation, therefore, is

IF ((FB is X1) AND (CO is X2) AND (EP is X3) AND (PF is X4) AND (UD is X5) AND
(ER is X6) AND (CL is X7) AND (PR is X8) AND (IN is X9) AND (HE is X10))

THEN $a_q0 + a_q1 * FB + a_q2 * CO + a_q3 * EP + a_q4 * PF + a_q5 * UD + a_q6 * ER + a_q7 * CL + a_q8 * PR + a_q9 * IN + a_q10 * HE$

The total number of possible fuzzy rules if we have n inputs and use K fuzzy sets to span the universe of discourse for each input is equal to K^n . For the usability problem, therefore, the total number of rules that we can have is $312 = 531441$.

This takes into consideration all possible combinations of the inputs. This is a rather extensive number of rules and many of these rules are redundant and hence will be deleted, in order to make the system more manageable. Further we note, that for any input instance, $212 =$ the maximum number of rules that could be triggered.

4.2 Tuning Algorithms for the Fuzzy Controller

As explained earlier, we need to tune the parameters that characterize the input membership function and the output crisp function associated with each rule. If we consider the most general form of the Takagi-Sugeno model

i.e. for rule q IF ($x1$ is $X1$) AND ($x2$ is $X2$) ...AND (xn is XN) THEN $g(x1, \dots, xn)$

Then the problem of tuning the parameters is illustrated in Figure 2 below.

The output associated with input example $\underline{x}^r$ is $y^r(\underline{x}^r)$ and this is given for the linear Takagi-Sugeno controller by expression of:

$$y^r(x^r) = \frac{\sum_{q=1}^M \alpha q^r \left(\sum_{s=1}^N \alpha q^s \times s^r \right)}{\sum_{q=1}^M \alpha q^r}$$

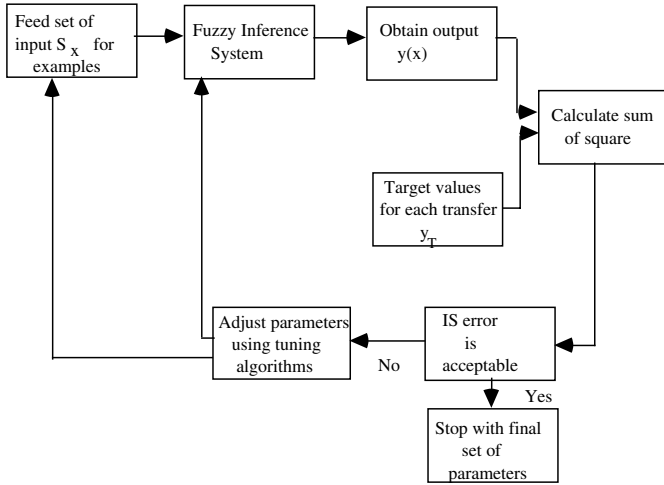


Fig. 2. Diagnostic Representation of Tuning Process

Here αq^r is the firing strength corresponding to rule q for the inputs x^r corresponding to the r . Let there be R examples altogether and let us write the target value of output corresponding to example r (or inputs x^r) as y_{Tr} . Then the sum of the squares of the errors generated will be given by expression below.

$$E(\underline{\rho}) = \sum_{r=1}^R y_{Tr} - y(x))^2$$

$\underline{\rho}$ is the at of parameters that will be adjusted and these consist of the parameters that characterize the input membership functions and the output coefficients.

$$\begin{bmatrix} \alpha_{10} \dots \alpha_{1N} \\ | \dots \dots \dots | \\ | \dots \dots \dots | \\ | \dots \dots \dots | \\ \alpha_{MO} \dots \alpha_{MN} \end{bmatrix}$$

The tuning algorithms that are available seek to minimize the value of this error function. There are many different tuning algorithms for doing these. We will only discuss the three main approaches to doing this and they are

- (i) Least Square Error Optimization Approach
- (ii) Gradient Method
- (iii) Neural Net Method

4.3 Fuzzy Usability Control Rules

The individual fuzzy sets, ie the usability factors described above, will determine the result of the overall usability of the system being tested. Each usability value for a particular set of usability factor values will be determined through the use of Fuzzy

Control Rules. If Feedback is good, Consistency is good, Error Prevention is poor,..., Error recovery is average then Usability is (some_value).

This is the rule in its simplest format. For the control rule to be intuitive there are a number of other factors which need to be considered. Each control rule has to be set up so that the output is some function of the inputs. The control rules are to be set up so that a linear relationship exists between the inputs and the outputs.

It is important to note that there are several control rules to properly model Usability. This is because a particular usability test may result in a particular set of usability factor values, and another particular instance may lead to a completely different set of parameter values. Both of these instances have to be modeled using separate control rules, in order to determine the corresponding value of the usability.

4.4 Output Function for Usability

As the fuzzy inference method used is the T-S model the output membership functions are singleton spikes, rather than fuzzy sets. Each count of the parameters FB, CO, EP, PF, UD, ER, CL, PR, IN, HE, constitutes a deficiency or inadequacy towards the corresponding usability measure. Hence the higher the count of these web usability measures, then the worse it is (the more likely it is to be situated in the 'Poor' region) which would indicate a poor usability.

However, the Web Usability output generated is the reverse of this, since the higher value of generated output the more optimum the web usability is. This is to keep the results of the Fuzzy Inference System intuitive, since one would expect a high value of usability for an interface to represent a good e-Learning system. The usability output generated will be a continuous value between 0-10, and so a value of zero would represent an extremely poor interface, whereas a value of ten would indicate that the UI has an outstanding usability.

4.5 Evaluation of the Validity of Generated Rules

As with the Mfs, the parameters of the fuzzy rules are generated through examination of the local features of the training data. Hence the fuzzy control rules would seem to produce a reliable measure for input cases that are exact or close to the training data vectors. Input values not in the original training set were run with the set of fuzzy rules and it was found that the fuzzy rules predicted the results well.

The following results show the usability values obtained by plugging in varying types of input vectors into the fuzzy system. The order of the usability parameters of the input vectors are [feedback, consistency, error prevention, performance, error recovery, user opinion].

Table 1 shows the input test cases derived from varying two different input-output training pairs obtained from testing the 1st Good interface. These input vectors are created by individually increasing and decreasing the value of each usability factor of the original training data vector.

Table 2 shows the input test cases derived from varying two different input-output training pairs obtained from testing the Average interface.

Table 3 shows the input test cases derived from varying two different input-output training pairs obtained from testing the Poor interface.

Table 1. Test cases for “Good” Web user interface

Test Cases	Input Vector [FB, CO, EP, UP, PE, UD, ER, HE]	Usability
1	[0, 0, 2, 0, 1, 1, 1, 0]	9.3625
2	[1, 0, 2, 0, 1, 1, 1, 0]	9.2780
3	[0, 1, 2, 2, 0, 3, 2, 0]	8.6695
4	[3, 1, 1, 1, 0, 0, 1, 2]	8.6847
5	[2, 2, 3, 0, 0, 2, 2, 1]	8.2833
6	[0, 2, 2, 3, 2, 1, 3, 2]	8.2870
7	[1, 4, 1, 1, 2, 2, 1, 2]	8.2975
8	[1, 2, 0, 2, 4, 4, 3, 2]	7.7440
9	[3, 3, 4, 2, 3, 3, 2, 1]	7.6687
10	[1, 4, 0, 2, 4, 4, 3, 2]	7.7779
11	[4, 3, 3, 2, 8, 4, 3, 1]	7.0687
12	[2, 1, 5, 4, 1, 3, 1]	7.0244
13	[0, 1, 1, 2, 2, 0, 0, 2]	9.0647
14	[1, 2, 2, 0, 0, 1, 1, 2]	8.9672
15	[0, 2, 1, 1, 0, 0, 2, 1]	8.8174

Table 2. Test cases for “Average” Web user interface

Test Cases	Input Vector [FB, CO, EP, UP, PE, UD, ER, HE]	Usability
1	[4, 6, 2, 3, 1, 3, 1, 4]	6.8594
2	[5, 3, 5, 2, 1, 3, 2, 3]	6.8755
3	[4, 3, 5, 2, 2, 3, 3, 3]	7.0626
4	[6, 2, 0, 7, 4, 4, 8, 2]	5.3373
5	[6, 4, 0, 7, 4, 4, 8, 2]	5.3379
6	[8, 9, 9, 7, 8, 8, 6, 8]	5.0759
7	[1, 4, 6, 2, 6, 1, 6, 6]	6.0988
8	[1, 4, 5, 2, 8, 2, 6, 2]	6.7517
9	[0, 5, 4, 5, 3, 2, 3, 3]	6.6173
10	[4, 3, 5, 2, 2, 4, 3, 5]	6.6264
11	[2, 2, 1, 4, 3, 6, 5, 3]	5.2078
12	[6, 6, 4, 5, 2, 1, 7, 3]	5.0684
13	[6, 6, 5, 5, 2, 1, 7, 3]	5.2720
14	[6, 6, 7, 1, 2, 1, 8, 3]	5.2414
15	[6, 6, 8, 1, 2, 1, 8, 3]	6.4825
16	[2, 3, 4, 4, 5, 2, 5, 3]	5.284
17	[7, 3, 5, 4, 6, 2, 1, 6]	6.5853

Table 3. Test cases for “Poor” Web user interface

Test Cases	Input Vector [FB, CO, EP, UP, PE, UD, ER, HE]	Usability
1	[8, 9, 4, 7, 8, 7, 6, 7]	2.7513
2	[8, 9, 6, 7, 8, 7, 6, 7]	2.4452
3	[4, 6, 10, 7, 10, 5, 8, 7]	2.9814
4	[4, 4, 10, 7, 10, 5, 8, 7]	2.9908
5	[8, 9, 8, 7, 8, 8, 6, 8]	2.0148
6	[8, 9, 9, 7, 8, 8, 6, 8]	1.9567
7	[2, 9, 10, 8, 9, 8, 6, 8]	2.6624
8	[5, 7, 5, 9, 8, 10, 10, 7]	2.7737
9	[5, 5, 5, 9, 9, 10, 10, 6]	2.6156
10	[10, 6, 10, 4, 4, 9, 9, 8]	2.6472
11	[10, 10, 6, 10, 8, 6, 7, 6]	2.1831
12	[6, 7, 10, 10, 10, 6, 7, 8]	1.9795
13	[6, 10, 7, 7, 7, 9, 10, 10]	1.7256
14	[10, 9, 10, 9, 9, 3, 4, 10]	1.6987
15	[9, 6, 10, 8, 8, 10, 10, 6]	1.7738
16	[10, 10, 9, 8, 6, 6, 7, 10]	1.5779
17	[9, 10, 8, 7, 9, 8, 10, 8]	1.2047
18	[7, 4, 9, 7, 10, 6, 7, 10]	2.4031

4.6 Discussion of the Results

The fuzzy system produces reliable measures for usability for input vectors that are close to the values of the training input-output pairs for each of the interfaces used in the testing. The fuzzy system is able to intuitively predict the direction the overall usability should go when particular inputs are increased and decreased relative to a particular training data vector.

When feedback, consistency, error prevention, performance, and error recovery are increased, the overall usability value generally decreases, which is what is intuitively expected. Also if these factors are decreased, then the overall usability is increased which is depicted in the above tables.

These results depict the importance that feedback and consistency have on the overall usability. When these two factors are increased and decreased, the usability significantly changes in the expected direction. In Table 1 when the feedback value in vector is changed from 6 to 3 the usability increases from 6.5000 to 9.0028. When the consistency value of test 4 is increased from 2 to 6, the usability decreases

considerably from 8.3482 to 3.5336. This test shows the importance that feedback and consistency have in the FIS at affecting the overall usability, which is representative of real-world results.

The usability factor of opinion also behaves in a consistent manner. Increasing and decreasing the user opinion affects the usability value in the same direction. These tables highlight the significant effect that opinion has on the overall usability. A reasonable increase or decrease in opinion changes the value of the overall usability considerably.

In Table 3, has a usability value of 6.9287, when the user opinion is changed from 6 to 10 the usability increases significantly to 9.1340. This is what is expected from the features of the training data. Of all the inputs, opinion is the one which resembles the desired usability rating across all training pairs. Hence, significantly altering the user rating will cause a similar change in the output value for usability.

5 Conclusion

A usability measurement approach for e-learning was discussed in this paper. We utilized some traditional usability measures and recent results based Usability Concepts [2, 14, 15, 16]. Data gathered on the actual e-learning system using Automated Usability Tool [13] was used to tune the parameters of the Fuzzy Model. The results obtained from the Automated Usability Software indicate that it is a suitable measure to e-learning.

References

1. Dillon T & Maquire, M. Usability Measurement- Its Practical Value to the Computer Industry. Proc. ACM/IFIP Human Factors in Computer System, INTERCHI 4, 1993 pp145–148.
2. Jakob Nielsen, "Designing Web Usability", 2000
3. Bavan, N. & Macleod, M "Usability measurement in context. Behaviour and Information Technology, (1994) pp132–145
4. Mamdani E & Assilian S. An Experiment Linguistic Synthesis with a Fuzzy Logic Controller. Int. J. Man Machine Studies 1–1 1975
5. Molich, R & Nielsen, J. Heuristic Evaluation of User Interfaces proceedings of ACM Human Factors in Computing Systems CHI'90 4, 1990 pp249–256
6. Nielsen, J. & Philips, V. Estimating the relative Usability of Two Interfaces: Heuristic, Formal, and Empirical Methods Compared. Proc. ACM/IFIP Human Factors in Computing Systems INTERCHI'93 4, 1993 pp214–221
7. Nielsen, J. Enhancing the Explanatory power of Usability Heuristics. Proc. ACM/IFIP Human Factors in Computing Systems INTERCHI'93 4, 1993 pp152–158
8. Takagi, T & Sugeo, M. Fuzzy Identification of Systems and Its Applications to Modelling and Control. IEEE Transactions on Systems man. & Cybernetics Vol SMC-IS, No 1 pp116–131
9. Thimbleby, H. Formulating Usability. SIGCHI Bulletin (APRIL 1994) pp59–64
10. Treu, S. User Interface Evaluation "A Structured Approach" 1994 pp 175–195
11. Shackel, B. The concept of usability, Proc. IBM Software and Information Usability symposium, Sept. Poughkeepsie, N 1981

12. Chang, E., and Dillon, TS Automated Usability Testing in Human Computer Interaction INTERACT97. pp77-84, Edited by S. Howard, J. Hammond, G. Linaard. (Book Chapter) Chapman and Hall London. 1997
13. Chang, E., Dillon, TS. And Cook, D. "An Intelligent System Approach to Usability Evaluation" (IIS'97). IEEE Conference on Intelligent Information Systems, Bahamas, 8-10 December 1997, pp218-226.
14. Chang, E., Dillon, T.S., Cook D. "Measurement of Usability of Software Using a Fuzzy System Approach" Invited paper Proceeding of Eighth International Conference on Software Engineering and Knowledge Engineering, pp69-76, Lake Tahoe, Nevada, USA
15. Wong, Siu Keung (Brian)., Chang, E., "Usability Measure For Web Based Applications". Software Metrics Conference, Melbourne, Australia, 2002.
16. Shackel, B.. The concept of usability. Proceedings of IBM Software and Information Usability Symposium, September 15-18, 1981, Poughkeepsie, New York, USA, 1-30. Poughkeepsie, NY: IBM Corporation.
17. Allison Rossett et. al., The ASTD ELearning Handbook: Best Practices, Strategies, and Case Studies for an Emerging Field, McGraw Hill, 2002.
18. Rutenbur, B., Spickler, G. and Lurie, S., E-Learning: The Engine of the Knowledge Economy, Morgan Keegan & Co., Inc. 2000.
19. Charles A. Shoniregun, Sarah-Jane Gray, Is E-learning really the future or a risk?, ACM Ubiquity, April 2003.
20. Karl M. Kapp, E-learning basics: essay: Five technological considerations when choosing an e-learning solution, ACM eLearn, June 2003.
21. Michael Feldstein, E-learning basics: essay: developing your e-learning for your learners, ACM eLearn, Sep 2002.
22. Susanna Tsai, Paulo Machado, E-Learning Basics: Essay: E-learning, online learning, web-based learning, or distance learning: unveiling the ambiguity in current terminology, ACM eLearn, Jul 2002.

Creating a “DOGMatic” Multilingual Ontology Infrastructure to Support a Semantic Portal

Jan De Bo, Peter Spyns, and Robert Meersman

Vrije Universiteit Brussel - STAR Lab
Pleinlaan 2, Gebouw G-10, B-1050 Brussels, Belgium
{Peter.Spyns,jdebo,meersman}@vub.ac.be
<http://www.starlab.vub.ac.be>

Abstract. We present extensions to the current DOGMA ontology engineering framework so that the DOGMA ontology server is better equipped to cope with context and multilinguality, issues that are important when deploying “ontology technology” in a setting involving human users. An obvious example is a semantic portal that offers search facilities to its users. A short presentation of the DOGMA ontology engineering approach will be provided, before the extensions mentioned will be presented.

Keywords: Ontology and database modelling, context, multilinguality

1 Introduction

More and more semantic portals, OntoWeb [33], KAON [23], OLR [7], or semantic based query methods, e.g. [5], are becoming popular on the internet or rather the semantic web. Our claim is that ontologies need to be linked with linguistic knowledge since the “eyeball web” with its human users is and will remain an important part of the semantic web (with its intelligent software agents). An important issue is the interaction with a user who wants to look up all kinds of information offered by a portal. A semantic portal provides for, amongst other things, the “visual input side” of a semantic retrieval engine¹ that is connected with an ontology server. Users can browse the underlying ontology to construct their query, which can become problematic with a large and complex ontology. The most easiest way for a human user to express a query is through natural language terms - as is the case for classical search engines.

The goal of this paper is to present some extensions on the current DOGMA (Developing Ontology-Guided Mediation for Agents) framework for ontology engineering [17] that are in with line the usage of natural language terms for semantic querying. We believe that the incorporation of “contexts” and “natural language terms” in an ontology is needed to correctly determine the meaning of a user query. The introduction of bridge types will allow to resolve issues about linking variables local to an application to ontology vocabulary.

¹ The presentation of the search results will not be considered here.

First, a general background of the field is given with a discussion of a specific semantic portal (see section 2). Since the underlying knowledge model of our ontology research is based on the DOGMA approach, we provide in section 3 an overview of the DOGMA initiative, a VUB STAR Lab research project on ontology engineering. Subsequently, we present a proposal for a more sophisticated treatment of contexts (section 4) and multilinguality (section 5). The introduction of bridge type definitions (see section 6) should allow to link information system applications to ontologies. Related work is discussed in section 7. Some indications on future work and final remarks (section 8) conclude this paper.

2 Background

Semantic portals today basically allow a user to define his/her query by means of structured input, i.e. a user selects terms (values) from drop down boxes that relate to properties of a concept in the underlying ontology. An important user interface problem is how to graphically represent a (large) ontology so that a user can easily browse and compose his/her semantic query (i.e. selecting the appropriate concept(s) and values). Several types of graphical representations of an ontology are studied, e.g. a tree [25], a hyperbolic view [34]:p.482] or a landscape view [30]. Sometimes a seemingly “regular” search box - e.g. the OntoWeb portal [25,33] - “hides” a semantic search box if the search is done over the meta-data (ontology-based annotations) instead of the “raw” data. Notice that a user has to be familiar with the vocabulary of the ontology (i.e. consisting of ontology terms, opposed to intuitive natural language terms).

The metadata² is stored by back-end ontology servers - e.g. Sesame based on RDF [2]. An overview of ontology servers in general is given in [6]. None of these has a thoroughly developed view on multilinguality. This means that there is no, or only a very superficial, difference between a linguistic term that lexicalises a concept and a logical term that represents a concept in an ontology. Synonyms and translations become problematic in such an approach. We believe that multi-lingual language terms should be linked to a concept that is uniquely labelled by a specific natural language head term or an artificially constituted concept label. By doing so, the conceptualisation achieves a high degree of language independence. The back-end ontology server should thus be designed and equipped accordingly.

At first sight, the KAON Portal is an exception as it is said to cope with homonyms (and synonyms ?) [23]. However, it is not so clear to what extent its “Lexical Ontology Instance Model” (LOIM) is actually integrated in the KAON portal³. Also, the use of “context” to disambiguate terms (see section 3.3) is not considered, or at least not further explained. Finally, the KAON method to “lift” databases to the ontology level [35] seems to be rather focused on lifting a single database of which the conceptual model largely determines the ontology, rather than providing techniques to allow multiple databases (or other applications) to

² The raw data remains at its original location.

³ The LOIM is not mentioned anymore in the remainder (examples) of the paper [23].

commit their internal model to a single domain ontology that is supposed to subsume individual applications or databases (see [32] for a related discussion).

3 DOGMA

Most often an ontology is defined as an *explicit, formal specification of a shared conceptualisation of a certain domain* [10], or as a *logical theory accounting for the intended meaning of a formal vocabulary* [13]. A DOGMA inspired ontology is based on the principle of a double articulation: an ontology is decomposed into an *ontology base*, which holds (multiple) intuitive conceptualisation(s) of a domain⁴, and a layer of *ontological commitments*, where each commitment holds a set of domain rules to define *a partial semantic account of an intended conceptualisation* [12]. The DOGMA approach of VUB STAR Lab is based on three rather evident observations that:

1. agreements become easier if the items involved are simpler
2. most relevant human knowledge is massively available in natural language, in text documents and other “lexical” sources such as databases
3. conceptualisations – and hence ontologies – should be as independent as possible of intended application and design(er) context, and of the language used to reach the agreement

As these requirements however are mutually counteracting except for the most trivial of domains, a heuristic approach is adopted based on the extensive practical experience of database technology under the *model-theoretic* perspective for relational databases [28]. As a summary, one can state that the DOGMA approach takes agreed semantical knowledge out of an application by making use of an external ontology (and achieves a form of “meaning independence”) [20]. This is done in much the same way that databases take data structures out of regular programs (“data independence”) – see [17,32] for details on DOGMA.

3.1 DOGMA Ontology Base

Currently, the ontology base consists of sets of intuitively plausible conceptualisations of a real world domain where each is a set of context-specific “representationless” binary facts types, called *lexons*, formally described as $\langle \gamma \text{ term}_1 \text{ role co} - \text{role term}_2 \rangle$, where γ denotes the context, used to group lexons that are logically related to each other in the conceptualisation of the domain [20]. Informally we say that a lexon is a fact that may hold for some application, expressing in that case that within the context γ the *term*₁ (or head) may plausibly have *term*₂ (or tail) occur in an associating *role* (with *co* – *role* as its inverse) with it. Lexons are independent of specific applications and should cover relatively broad domains. Lexons in a DOGMA ontology base are always “true”, i.e. free of further interpretation. E.g., “*bookstore: book is_identified_by/identifies ISBN*” is a lexon, with “bookstore”= γ , “book”= head, “ISBN”= tail, “is_identified”= role and “identifies” = co-role.

⁴ Guarino would call this an *uninterpreted ontology*.

3.2 DOGMA Commitment Layer

The commitment layer, mediating between the ontology base and applications, is organised as a set of ontological commitments [11], each being an explicit instance of an (intensional) first-order interpretation of a task in terms of the ontology base. A commitment is a consistent set of rules (or axioms) in a given syntax that specify which lexons of the ontology base are visible (*partial account*) for usage in this commitment and that semantically constrain this view (i.e. the visible lexons). The rules that constrain the relations between the concepts (*semantic account*) of the ontology base are specific to an application (*intended conceptualisation*) using the ontology. Experience shows that agreement on the domain rules is much harder to reach than on the conceptualisation [21]. E.g., it is easy for people to agree on the binary fact represented by the lexon “*bookstore: book is_identified_by/identifies ISBN*” in a “bookstore” context, while they might disagree whether or not for a *given* application the ISBN number is a *mandatory* property of a book. An application (e.g., semantic portal) can adopt more than one commitment. A commitment, in principle, can be shared by several applications. Commitments can also include derivation and inference rules. Sets of ontological commitments can be regarded as reusable knowledge components (e.g., several applications can use the same “search commitment rules”).

3.3 DOGMA Contexts

Contexts have the generic property of disambiguating the lexical meaning of terms inside a lexon. Until now a context is represented by a symbol $\gamma_i \in \Gamma$, where Γ is the context space of the domain to be modelled. Currently, γ_i is a mere label that refers in a non formal way to a source (e.g., a document that contains and “explains” how the various terms are used in that particular context).

Two terms $t_1 \in T$ and $t_2 \in T$ are synonyms when they identify the same concept within the same context γ . When the meaning of identical terms is different, we speak about homonyms. For example, the term *bank* has a different meaning in a *geographical* context than in an *financial* context and therefore evokes different concepts. We refer to the classical semiotic triangle [24]: a word or term is a symbol in natural language evoking a language-independent concept in our mind that refers to a world object (referent). A term only indirectly (via the corresponding concept) “points” to a referent. The “evokes” relationship between a term and a concept has a cardinality of m:n.

4 Extensions to DOGMA

4.1 Extension of the Context Definition

We now redefine a context $\gamma_i \in \Gamma$ as a semantic cluster of concepts that are logically and meaningfully related. It is the task of the ontology engineer to build these contexts. To establish a relationship between terms and concepts in a given context γ_i , we define a *context mapping* ψ_i , from a domain T (the set of terms)

to a range C (the set of concepts within that particular context γ_i), formally noted as $\psi_i : T \rightarrow C$, so that $range(\psi_i) = \gamma_i$. This is a m:1 mapping, because several terms can be associated with the same concept. A context mapping ψ_i thus determines the meaning of terms by establishing a formal link between lexical terms and its underlying concept in a certain context γ_i . By definition the following equation holds $\Gamma = \bigcup_i \gamma_i = \bigcup_i range(\psi_i)$.

4.2 Representation of Concepts

According to the DOGMA approach, terms are part of a *lexon* and are represented by natural language *words* in the ontology base. To describe a concept we propose to associate with each concept a *set of synonymous terms*. The idea of using a *synset* (=set of synonyms) to express the semantics of a concept is inspired on the approach taken in WordNet [8].

Wordnet offers two distinct functionalities: a *vocabulary* to disambiguate terms (=to describe the various senses of a term) and an *ontology* to describe the semantic relationships among the various senses. Wordnet can be seen as an early precursor of ontology development: it includes semantic relations among the synsets in the ontology of Wordnet as hyperonymy, hyponymy, meronymy next to antonymy and value_of (referring to the noun database of WordNet.)

An example will clarify: the natural language **term** “*java*” can “point to” different meanings according to the specific context it is used in. If we use it in the context of “*Computer Science*”, the term⁵ “*java*” refers to a platform independent, object-oriented programming language. If it is used in the context of “*travelling*” it refers to an island in Indonesia and when used in the context of “*consumer goods*” it refers to coffee. The concepts lexicalised by the term *java* ($t = \text{“java”} \in T$) will be denoted as follows in the respective contexts:

$$\begin{aligned}\gamma_1 &= \text{“CompSc”}; \psi_1(t = \text{java}) = c \equiv \{\text{“java”}, \text{“programming\_language”}\} \\ \gamma_2 &= \text{“ConsGoods”}; \psi_2(t = \text{java}) = c' \equiv \{\text{“java”}, \text{“coffee”}, \text{“beverage”}\} \\ \gamma_3 &= \text{“Travel”}; \psi_3(t = \text{java}) = c'' \equiv \{\text{“java”}, \text{“island”}, \text{“vacation\_destination”}\}\end{aligned}$$

It is our aim to describe concepts by defining them as a set of semantically equivalent terms. The equivalence sign “ $\equiv$ ”, is used to describe the semantics of a concept.⁶

Formally we state that: $\psi_i(t) = c \equiv \{t, t', t'', t'''\}$, where $t, t', t'', t''' \in T$ and $c \in \gamma_i$.

This specification allows a machine to retrieve, compare etc. concepts. These unique combinations of synonymous terms describe the logical vocabulary we use

⁵ One has to remember the difference between a natural language or lexical term and a logical term (or ontology term), which sometimes in the AI literature appears as a synonym of concept.

⁶ It is thus obvious that the lexical term “java” is included in the set of synonymous lexical terms describing the related concept, which might also be labelled by the string “java”. The latter is a logical term while the former is a natural language term - see the previous footnote

to model the given domain. Because ontology engineering often concerns rather specific domains (e.g. complaint management, European directives, surgery) to be modelled, we cannot only rely on Wordnet's vocabulary since it exclusively includes the 95.000 most common English words and lacks very specific or technical terms. Therefore we prefer to build our own vocabulary, separated but nevertheless compatible with and building on Wordnet.

Besides grouping semantically equivalent terms in order to uniquely and unambiguously describe the concepts occurring in the ontology, additional phrases (explanatory glosses) are needed to clarify a meaning or a concept. A gloss can contain a natural language description and/or a formal language definition (e.g. using Conceptual Graphs [31]). One can consider the gloss as the definition by intension. In practice, the natural language gloss will often be taken from one of the resources the domain expert has at his/her disposal when modelling the domain. In the realm of the previous example and considering the context of "*Consumer Goods*", a gloss for the concept c' represented by the term $t = \text{"java"}$ could be "*a beverage consisting of an infusion of ground coffee beans; he ordered a cup of coffee*" [38].

4.3 Representation of Context Space

In later work based on the pioneering research of McCarthy [19], Buvac concludes that common sense knowledge is needed to completely *lexically disambiguate* a proposition like *Vanja is at a bank* [4]. As a consequence, trying to resolve lexical ambiguity using a formal theory for context like McCarthy and Buvac propose does not seem to be the most practical way, at least for our purposes, to deal with the phenomenon of context. Indeed, it would require a human to explicitly add common sense knowledge to the reasoning system to solve a lexical ambiguity.

Since we have redefined a context ($\gamma_i \in \Gamma$) as the range (set of values) of one particular context mapping ψ_i , it can be formally interpreted as a particular (i.e. meaningfully related) set of concepts, which on their turn are labelled by a specialised domain vocabulary and explanatory glosses. Note that the idea of the "corpus based word relator" of Mitra and Wiederhold [22] is a similar attempt to disambiguate terms (in view of ontology integration), although they did not make the difference between a language term and an ontology term or concept (see section 6.3).

Our current thinking is to organise all these contexts ($\forall \gamma_i \in \Gamma$) in a tree structure. Each node in this tree thus represents one particular context γ_i . Because each separate context is a set of meaningfully grouped concepts, the context tree is populated with concepts. The tree is layered starting from the top with the 0^{th} level. This level contains the most common contexts of the context space. The 0^{th} level is then expanded by the contexts of the first level. Each context of the first level is derived from a context of the 0^{th} level, which means that the concepts of the first level context supplement and further refine (but not replace) the concepts of the 0^{th} level context. Let us assume that γ_i^1 , a context from the first level, is derived from γ_j^0 , a context from the 0^{th} level in the context tree. The set of concepts of γ_i^1 expands the set of concepts of γ_j^0 . Formally we state

that, $\gamma_j^0 \subset \gamma_i^1$. The same logic holds for all the other levels in the tree. We can thus generally state that if a context γ_x^{n+1} of the $(n+1)^{th}$ level is derived from a context γ_y^n of the n^{th} level, then $\gamma_y^n \subset \gamma_x^{n+1}$.

Context trees may be helpful to describe the precise and unambiguous meaning of terms. This may be desirable for various reasons. For instance, a domain expert may want to determine the unambiguous meaning of a term, which he found in a resource describing a particular domain. It may be the case that a lot of words (i.e. terms) in the close environment of that particular term determine its context. This will often lead to a specific context node in the context tree since a context is a set of logically related concepts. If this context is not specific enough to determine the precise meaning of the term, the domain expert will have to descend one level in the tree (e.g. from level n to level $n+1$). If the meaning of the term is not found at all, the ontology engineer will have to expand the context tree at the node where the search activities stopped. In other cases (e.g. ontology integration) the tree will be more likely used by a software agent instead of a domain expert. We visualise the idea of a context tree by means of the example depicted in Figure 1.

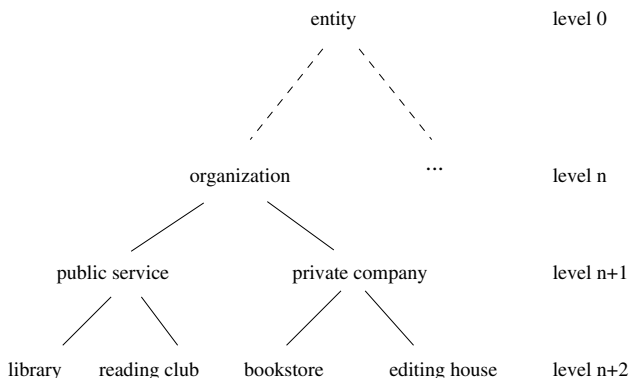


Fig. 1. Visualisation of a context tree

Note that the context tree is by no means complete. Real context trees would probably contain more subtrees. Also for spatial considerations, the figure does not contain any accompanying concepts for the contexts being represented. In figure 1 the *library* context extends the *public service* context with additional concepts.

It is our intuition that one could reuse (parts of) an upper level ontology to build a context tree. Contexts can be seen as a more general and coarser way of structuring a conceptualisation. As a consequence, a context might be equivalent to a certain extent with upper ontologies. However, in our approach, a context tree remains a separate entity (not merged or aligned with the ontology

proper). For time being, we stick to a tree structure. Future work will provide an indication on the necessity of transforming the tree into a lattice.

5 Multilinguality

The DOGMA meta-model does not yet foresee in a multilingual representation of concepts. Until now we assumed the modelling language was English and used a term t to label a concept c . However, when mining resources in order to build an ontology one cannot expect that all these resources occur in one and the same language. A classical, non technical, example is the following: in the context of “smoking”, the French term “*feu*”, the English term “*light*” and the Dutch term “*vuur*” identify the same concept. In the context of a house, the English term “light” is translated in French as “*lumière*” and “*licht*” in Dutch. The Dutch term “*vuur*” corresponds to the French “*feu*” that is translated in English by “fire”. As one can see in Table 1, not all notions are lexicalised in a language by a different term, and the “meaning distribution” as a result of the lexicalisation process is not the same for all languages. Other similar examples can be found in [29]:p.84]. Therefore we want to introduce a new *linguistic identifier*, called $\lambda \in \Lambda$, where Λ is the linguistic space.

Table 1. Example of a different meaning distribution for translated terms

	<i>English</i>	<i>French</i>	<i>Dutch</i>	
γ_1	<i>fire</i>	<i>feu</i>	<i>vuur</i>	$\rightarrow c_1$
γ_2	<i>light</i>	<i>feu</i>	<i>vuur</i>	$\rightarrow c_2$
γ_3	<i>light</i>	<i>lumière</i>	<i>licht</i>	$\rightarrow c_3$

Now we can extend our definition of a *concept* as introduced above in section 3.3. In a given context $\gamma \in \Gamma$, any concept can be represented by multiple terms in multiple natural languages. Each natural language corresponds with an identifier $\lambda_i \in \Lambda$. In our example, λ_1 corresponds to Dutch, λ_2 corresponds to English and λ_3 corresponds to French.

With a given context $\gamma_j \in \Gamma$, which is equal to “smoking” in our example, we associate the context mapping ψ_j . We now define: $\psi_j \lambda_i t_i \rightarrow c$ with $\lambda_i \in \Lambda$. In our example holds, t_1 =“vuur”, t_2 =“light” and t_3 =“feu”. These terms are lexical representations in different languages of the concept c that we will denote as follows in our vocabulary: $c=\{\{\text{“lighter”}, \text{“light”}, \text{“igniter”}, \text{“ignitor”}\}; \text{a device for lighting or igniting fuel or charges or fire; “Do you have a light?”}\}$. We hope to apply the same mechanism to conceptual relationships.

From the above, it follows that lexons are a necessary but intermediary (since language dependent) representation. It echoes Guarino’s statements that currently in AI the term “ontology” collapses the language independent conceptualisation level with the language dependent ontology level [13]:p.8]. Of

course, we have to understand the term “ontology” in the way Guarino has circumscribed it - see [12]. A DOGMA ontology base (or rather “a conceptualisation base”) eventually should consist of binary “conceptons”⁷ formalised as (*concept₁ relation co – relation concept₂*) instead of lexons⁸. As a corollary, the commitment layer puts constraints on the conceptons and thus gains in scope. However, more research on this topic within the DOGMA framework is needed (e.g., how does multilinguality affect a context tree: is it enough to include the terms of other languages in the set of terms that define a concept?).

6 Bridging the Semantic Gap between an Information System and a DOGMA Ontology

6.1 NIAM Bridge Types

In order to restrain the ontology base (basically selecting lexons), applications (information systems) select sets of particular commitments, stored in the commitment layer. To express these links, we use *bridge types*. This terminology is taken from an information systems modelling methodology called aN Information Analysis Method (NIAM [37]), that evolved later to Object Role Modelling (ORM [14]).

In NIAM, bridge types are defined as binary fact types between lexical objects and non-lexical objects. We adopt the NIAM philosophy of a strict separation between lexical objects (LOTs) and non-lexical objects (NOLOTs). A lexical object or label is *an object in a certain reality which can be uttered, written down, or otherwise represented* [o.c.]. LOTs always consist of letters, numbers, symbols or other characters. They can be used as names for or references to other objects. A non-lexical object (NOLOT) is *an object in a certain reality which cannot be uttered, written down or otherwise represented. Non-lexical objects must be named by lexical objects or referred to by means of lexical objects*. [o.c.].

6.2 Bridge Types from the Information Systems to the Ontology Level

In practice, establishing the bridge between LOTs and NOLOTs explicitly will often boil down to a manual exercise. As an example, there could exist a bridge type between “isbn_no” (LOT) and “isbn” (NOLOT), “pub_date” (LOT) and “publishing_date” (NOLOT), “salary” (LOT) and “salary_amount” (NOLOT), etc. We also note that because the fields of database tables are mostly arbitrarily named, it will be very hard to automate the activity of establishing bridge types.

⁷ Other denominations could be meta-lexon or conceptual lexon.

⁸ We temporarily discard the issue of how to label a concept, e.g. numerical identifier vs. specific compositional artificial language expression vs. preferred natural language terms.

6.3 Bridge Types from the Ontology to the Conceptualisation Level

For the same reasons as we have introduced bridging types on the information system level, we will now apply this *bridging-logic* on the ontology-level (term-concept relation that we have introduced in section 3.3). We have defined the combination of a context mapping (ψ_i) and languages identifiers ($\lambda_i \in \Lambda$) as a mapping between terms (LOTs) and concepts (NOLOTs). Therefore this mapping forms a bridge type between lexicals and non-lexicals on the ontology level.

A very important note has to be made regarding the dual interpretation of a term in the ontology base. In the context of an information system, terms fulfill the role of NOLOTs whereas in the context of ontologies terms fulfill the role of LOTs because they are the lexical representations of the concepts (NOLOTs) in the ontology base. We visualised this distinction by means of Figure 2 that integrates through a flattened semantic triangle an information system and an ontology. The real world objects (referents) that are included in the information system can only be referenced by linguistic symbols (e.g., a term as a column name). However, as the semantics of these terms is implicit and therefore impossible to be shared amongst computers, the exact and shareable semantic definition (or intended meaning) of these terms (which can be different for each application - consider PN, PersName, PersonName, Name, Naam, Nom, ...) is defined on the logical level. Bridging between terms and concepts is done in the way explained in section 3.3. The same rationale is applied in case of terms in another natural language (see section 5).

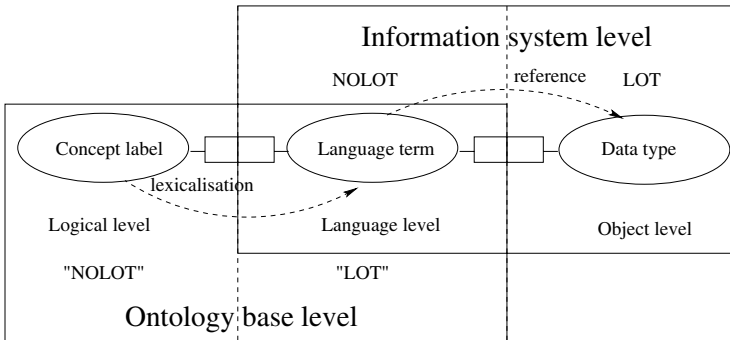


Fig. 2. Schema Overview

Perfect 1:1 matches (via bridges to the same concept) between terms of different languages and/or applications will not always be possible (e.g., see Table 1). Therefore, it is expected that specific equivalence or mapping rules will have to be defined that are typical of a particular application (e.g., “pub_date” → “publishing date”), and as such, not re-usable by similar applications. These rules are located in a separate and new layer: *the application layer*. The commitments, as they are defined on the logical level, are independent of the local vocabularies

- implying that LOTs are not allowed in a commitment - and thus in principle re-usable by similar applications⁹. This is depicted in Figure 3.

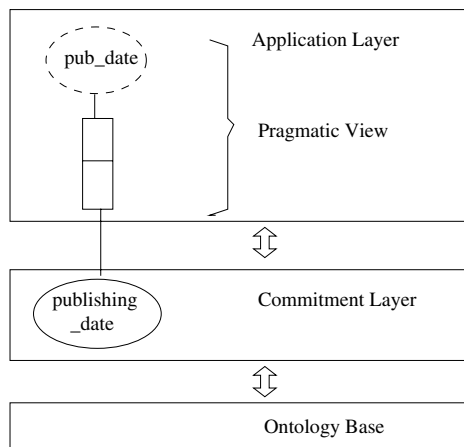


Fig. 3. Existing DOGMA architecture extended with an Application layer

7 Related Work

In addition to the related work already mentioned in section 2, we mention in this section other efforts on particular aspects of related to the work presented here. In [9], it is shown how contextual reasoning can be formalised as deduction in a system allowing multiple first order theories. Within the BUSTER project [3], a context theory is defined as a collection of linguistic expressions, providing an explicit description of the domain. Another approach on contexts was taken by Lenat [18] who distinguished 12 categories in a context space. He incorporated this idea in Cyc, the world’s largest knowledge base at this moment.

Pazienza and Vindigni [26] suggest to use natural language terms related to concepts to achieve an agreement on a shared representation of a domain. The context in which the terms are used helps to select the correct concept. One of the intended uses of the DOGMA ontology server is to function as a mediator. As such, the topics described in this paper blend in with the research on research on data mediators in general - e.g. see [1,36]. Mediator systems are presented in [27].

⁹ Although it could be possible to use LOTs inside of a commitment (implying the inclusion of the application layer in the commitment layer), a loss of reusability perspectives of a commitment is the net result since a commitment is now linked to the local vocabulary of a specific application.

Bridge types are well known in the ORM data modelling field, but have not yet been introduced in the field of ontology engineering before, at least to our knowledge.

Computational lexicographers are now also turning to the semantic web, as is illustrated by the creation of the MILE lexical model by the ISLE group, e.g., see [16]). Lexical semantics constitutes the natural complementary research area to be combined with research on ontologies in view of linking local terms of a conceptualisation to natural language terms .

8 Future Work and Conclusions

In this paper we have proposed some extensions to the existing DOGMA framework. We also stressed the multilingual aspect of ontologies. The next step is to redefine the meta-model of the DOGMA ontology server and implement the additions and refinements mentioned. Practically speaking, this will result in a powerful ontology server that combines conceptual with multilingual lexicographic/terminologic information. It is our intention to replace the parts of the OntoWeb semantic portal for which VUB STAR Lab is partly responsible [25] by this new ontology/lexicon server combination to allow for genuine semantic underpinned multilingual term based searching. Of course, once language terms are linked to ontology concepts, a plethora of natural language processing techniques and modules (e.g., robust parsers, dialogue modules, multimodal input, ...) can be combined to create genuinely intelligent information providing agents and applications offering an intuitive and user friendly way of interaction.

Acknowledgments. A major part of this work has been funded by the IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders): Jan De Bo has received an IWT PhD grant (IWT SB 2002 #21304) while Peter Spyns is supported in the context of the OntoBasis project (GBOU 2001 #10069). We would also like to thank our colleagues at STAR Lab for their stimulating discussions during the “Dagstuhl sessions”, as well as the reviewers.

References

1. Bergamaschi S., Castano S., Vincini M. & Beneventano D., (2001), Semantic integration of heterogeneous information sources, in *Data & Knowledge Engineering* 36 (3): 215–249
2. Broekstra J., Kampman A., & van Harmelen F., (2002), Sesame: An Architecture for Storing and Querying RDF and RDF Schema, In Proc. of the First Internat. Semantic Web Conference (ISWC 2002), Springer LNCS 2342, pp. 54–68
3. Bremen University Semantic Translator for Enhanced Retrieval.
<http://www.semantic-translation.de>
4. Buvac S. (1996). Resolving Lexical Ambiguity using a Formal Theory of Context. in Van Deemter and Peters (eds.), *Semantic Ambiguity and Underspecification*, CSLI Publications, Stanford

5. Chiang R., Eng Huang Chua C. & Storey V., (2001), A smart web query method for semantic retrieval of web data, *Data & Knowledge Engineering* 38: 63–84
6. Ding Y., Fensel D., Klein M. & Omelayenko B., (2002), The Semantic Web: Yet Another Hip ?, *Data and Knowledge Engineering* 41 (3): 205–227
7. Dhraief H., Nejdl W. & Wolpers M., Open Learning Repositories and Metadata Modeling, in *Proc. of the first Semantic Web Working Symposium*, 495–514
8. Fellbaum, C. (1998). Wordnet: An Electronic Lexical Database. Cambridge, US: The MIT Press.
9. Giunchiglia, F. (1993). Contextual Reasoning. *Published in Epistemologia : rivista italiana di filosofia della scienza*, 1993, vol: 16
10. Gruber, T.R.(1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2).
11. Guarino N., Carrara M., & Giaretta P., (1994), Formalizing Ontological Commitments, in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI94)*, AAAI Press pp. 560–568
12. Guarino N. and Giaretta P., (1995) Ontologies and Knowledge Bases: Towards a terminological clarification, in N. Mars (ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press A’dam, pp. 25–32
13. Guarino N., (1998), Formal Ontology and Information Systems, in N. Guarino (ed.), *Formal Ontology in Information Systems (Proceedings of FOIS 98)*, IOS Press, Amsterdam, pp. 3–15
14. Concept Schema and Relational Database Design, Prentice Hall Austria, 1995
15. Heflin J. & Hendler J., (2000), Workshop on Artificial Intelligence for Web Search, Papers from the AAAI Workshop. WS-00-01, pages 35–40. AAAI Press
16. Ide N., Lenci A., & Calzolari N., (2003), RDF Instantiation of ISLE/MILE Lexical Entries, in *Proceedings of the ACL 2003 Workshop on Linguistic Annotation: Getting the Model Right*
17. Jarrar, M., Meersman, R. (2002). Formal Ontology Engineering in the DOGMA Approach. in Meersman R., Tari Z. et al. (eds) *On the Move of to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings*, LNCS 2519, Springer Verlag, pp. 1238–1254,
18. Lenat D. (1998). The dimensions of context-space, Austin, TX: Cycorp Report
19. McCarthy J. (1993). Notes on formalizing context. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*
20. Meersman, R. (2001). Reusing certain database design principles, methods and techniques for ontology theory, construction and methodology, *STAR Lab Technical Report*, <http://www.starlab.vub.ac.be/publications/STAR-2001-01.pdf>
21. Meersman R., (2002), Semantic Web and Ontologies: Playtime or Business at the Last Frontier in Computing ?, in *NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises*, pp.61–67.
22. Mitra P. & Wiederhold G., (2002), Resolving Terminological Heterogeneity in Ontologies, in *Proceedings of the Workshop on Ontologies and Semantic Interoperability at the 15th European Conference on Artificial Intelligence (ECAI 2002)*
23. Motik B., Maedche A. & Volz R., (2002), A Conceptual Modeling Approach for Semantics-driven Enterprise Applications, in Meersman, R. & Tari, Z., (eds.), in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings*, of series LNCS 2519, Springer Verlag
24. Ogden & Richards (1923), *The Meaning of Meaning*. 8th Ed. New York, Harcourt, Brace & World, Inc

25. Oberle D., & Spyns, P., (2003), The Knowledge Portal “OntoWeb”, in Staab, S. & Studer, R. (eds.), *Handbook on Ontologies in Information Systems*, LNCS, pp. 521–540, Springer Verlag [in print]
26. Pazienza M.-T. & Vindigni M., (2002), Language-based agent communication, in *Proceedings of the International Conference Knowledge Engineering and Knowledge Management Workshop on Ontologies for Multi-Agent Systems* [<http://www.csc.liv.ac.uk/valli/Papers.html>]
27. Rahm E. & Bernstein Ph., (2001), A survey of approaches to automatic schema matching, *The VLDB Journal* 10: 334–350
28. Reiter R., (1988), Towards a Logical Reconstruction of Relational Database Theory, in Mylopoulos J. & Brodie M.L., *Readings in AI and Databases*, Morgan Kaufman.
29. Resnik Ph. & Yarowsky D., (1997) A Perspective on Word Sense Disambiguation Methods and their Evaluation, in M. Light, *Tagging Text with Lexical Semantics, why, what and how ? (Proc. of the ACL – SIG on the Lexicon Workshop)*, pp. 79–86
30. Sabol V., Kienreich W., Granitzer M., Becker J., Tochtermann K., & Andrews K., (2002), Applications of a Lightweight, Web-based Retrieval, Clustering, and Visualisation Framework. in Karagiannis D. & Reimer U. (eds.), *Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM02)*, pp. 359–369, LNAI 2569, Springer Verlag
31. Sowa, J.F., (2000), *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co.
32. Spyns, P., Meersman, R., & Jarrar M. (2002). Data Modelling versus Ontology Engineering. in Sheth A. and Meersman R. (eds), *SIGMOD Record Special Issue on Semantic Web, Database Management and Information Systems* 31(4):12–17
33. Spyns, P., Oberle, D., Volz, R., Zheng, J., Jarrar, M., Sure, Y., Studer, R. & Meersman, R., (2002), OntoWeb – a Semantic Web Community Portal. Karagiannis, D. & Reimer, U., (eds.), in *Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM02)*, LNAI 2569, page 189–200. Springer Verlag.
34. Staab S., Angele J., Decker S., Erdmann M., Hotho A., Maedche A., Schnurr H.-P., Studer R., & Sure Y., (2000), Semantic community web portals, in *Proceedings of the 9th International World Wide Web Conference*, pp. 473–491, Elsevier
35. Stojanovic L., Stojanovic N. & Volz R., (2002), A reverse engineering approach for migrating data-intensive web site to the Semantic Web, in *Proceedings of the Conference on Intelligent Information Processing, World Computer Congress*, Kluwer Academic Publishers.
36. Wiederhold G., (1994), Interoperation, Mediation, and Ontologies, in *Proceedings International Symposium on Fifth Generation Computer Systems (FGCS94), Workshop on Heterogeneous Cooperative Knowledge Bases*, vol. W3, pp. 33–48, ICOT, Tokyo, Japan
37. Wintraecken J.J.V.R. (1990). *The NIAM Information Analysis Method, Theory and Practice*. Kluwer Academic Publishers, The Netherlands.
38. <http://www.cogsci.princeton.edu/wn/> [Wordline 1.7.1 on-line search]

Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints

Dirk Draheim and Gerald Weber

Institute of Computer Science, Freie Universität Berlin
`draheim@inf.fu-berlin.de`

Abstract. Form-Oriented Analysis is an approach tailored to the modeling of systems with form-based, submit/response style interfaces, a distinct and ubiquitous class of software systems. Form-Oriented Analysis models the system interface with a bipartite finite state machine and relates it to a layered data model. The paper explains the main visual artifact of our technique, the form chart, and introduces dialogue constraint writing. Model decomposition is explained. The analysis technique is firmly based on existing well-understood analysis notions and techniques, and consequently extends these methods.

1 Introduction

In this paper we present Form-Oriented Analysis, a new analysis technique for a distinct and ubiquitous class of interactive software systems. This class covers well-known form-based applications ranging from typical Internet shops through supply chain management to flight reservation systems. We give a precise definition of the considered class of software systems and have coined the term *submit/response* style applications for this system class.

Submit/response style applications are such applications that present to the user a page that offers information as well as a number of interaction options, typically forms. If the user has filled out a form and *submits* the form the system processes the data and generates a *response* page. This response page again offers different interaction options to the user. We model such a submit/response style application in a way that will turn out to be well suited for such systems, namely as a bipartite state machine, which alternates between presenting a page to the user and processing the data submitted by the user. This bipartite state machine is depicted in the key artifact of Form-Oriented Analysis, the form chart. Form-Oriented Analysis describes then, how to annotate this bipartite state machine with constraints, which specify the behavior of the system. The definition submit/response style is not intended to cover all kinds of software systems, but to single out a well-defined and important class of systems. There are of course other interactive software systems that do not follow this metaphor. In many software systems such as text editors or drawing tools the interaction with the system does not proceed by submission of forms that lead to a new page. Instead, the current screen is constantly updated as the interaction proceeds. However,

submit/response style applications form a ubiquitous class of important systems, which justifies the development of an analysis method specifically designed for this type of systems.

Technically, submit/response style applications can appear as modern web applications or as client/server applications or of course as classic mainframe applications. However, we deal in this paper with the *analysis* of such systems, and the particular technical representation shall be transparent for the functional specification as a result of the analysis phase, hence we want to achieve a *specification* independent from *implementation*. It is therefore the key goal of this paper to establish a high level view on this type of systems, in which we abstract from the underlying technology and focus on the inherent properties of submit/response style systems. Not every problem is amenable to a solution by a form-based system. But if a system is intuitively thought of as being accessible by a submit/response style interface this gives an important starting point for the problem analysis. In the analysis technique proposed here, called Form-Oriented Analysis, we will give a powerful approach to system modeling by understanding the system along its usage through a submit/response style interface. This interface model in Form-Oriented Analysis is an abstract interface; it is a conceptual tool for the understanding of the system. But it can be thought of as a working prototype of the actual application interface. Hence Form-Oriented Analysis is a technique for modeling a system along a prototypical submit/response style interface.

The restriction of Form-Oriented Analysis to submit/response style applications allows us to employ the clear semantics of submit/response style interfaces within the analysis phase. Hence a model obtained in form-based analysis benefits in its formal strictness and semantic clarity from the restriction to this interaction style. Form-Oriented Analysis covers the area of analysis which is typically called the functional specification.

Form-Oriented Analysis uses mainly visual artifacts for modeling. But in contrast to other visual modeling techniques we understand these artifacts mainly as a visualization of information, which also could be given in a textual representation. This flavor of visualization is important for Form-Oriented Analysis since it is a technique designed for tight integration into a suite of code based tools.

This paper explains the form chart, which is the main contributed artifact of Form-Oriented Analysis. The other form-oriented diagram types, page diagram and form storyboard [9], are important informal predecessors of the form chart, which highlight specific aspects. Page diagrams offer a natural conceptual basis for modeling of submit/response-style software system. Form storyboards are designed with respect to informal communication between domain experts and system analysts. The special way that signatures of server actions are visualized as forms make form storyboards able to serve as high-level prototypes. Form storyboards can be transformed into form charts without structural friction. Form charts are used for rigorous software system specification. Our method

offers a simple yet powerful composition mechanism for artifacts, called feature composition.

Form-Oriented Analysis does not prescribe any process model. Of course, the different degree of formality of the different artifacts as well as the feature composition mechanism hints at a process like intuitive succession of diagrams from informal to formal, but it is important to realize that this is by no means necessary. Since the semantics of the diagrams is completely independent from any process definition, the diagram is basically neutral with respect to its use in a process of whatever kind. However, our method can be used easily with agile methodologies due to similarity between form charts and the actual code architecture. With the feature composition paradigm, form charts can easily cope with changing requirements.

In section 2 we present the central artifacts of Form-Oriented Analysis. Dialogue constraints are discussed in section 3. Composition of artifacts is addressed in section 4. Tool-support for Form-Oriented Analysis is described in section 5. We give a dedicated discussion on selected related work in section 6.

2 Form Charts and Model

Form charts introduce into the system model one of the major conceptual cornerstones of Form-Oriented Analysis: the system is seen as a bipartite state transition diagram. The bipartite state machine is the basic structure of form charts. In this view the system is alternating between two kinds of states. The first kind of states corresponds to the pages of the system. These states are called client pages. The system remains in such a client page state until the user triggers a page change. In that moment the record with her previous input is sent to the system. The second kind of states represent the system actions in response to page changes and are therefore called server actions. These states are left automatically by the system and lead to a new client page.

We demonstrate Form-Oriented Analysis for an example seminar online registration system. Our example system is a form-based seminar registration system as it is suited for a single course. The screen shots of the system are shown in Figure 1. The front page shows the students registered so far and contains links to the different interaction options. New students can register themselves. From the homepage, students already registered can change or delete their registration. Each link leads to a new page specific for the chosen option.

The form chart has the task to make the analysis model amenable to formal constraint writing and coupling to the semantic data model, and it is therefore accompanied by two other diagrams, first the semantic data model and second the data dictionary mediating between both. Furthermore a textual document containing formal constraints has to be seen as attachment to the form chart. The document bundle consisting of form chart with attached constraints, data dictionary and semantic data model comprise the *form-oriented specification* of the system. A complete specification of a system is often a valuable goal, but in many cases it may not be practically achievable. Our method allows the

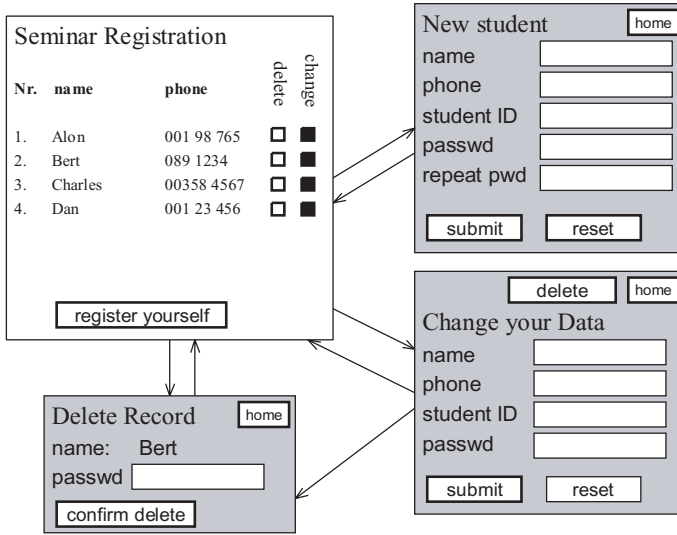


Fig. 1. Overview of the screens of an online seminar registration system.

modeler to create a complete specification, but of course it is usable for partial specification as well and therefore gives the modeler the freedom to choose the degree of precision which seems appropriate for the project.

The form chart as shown in Figure 2 contains the bipartite state machine. Server actions are depicted as rectangles and client pages are depicted as bubbles. In the form chart only the names of the states and transitions appear. The form chart also contains the start marker. The second new artifact type, the data dictionary, is shown in Figure 3. The data dictionary contains types and is therefore a class diagram in the terms of modern modeling languages like the UML. However, the data dictionary types are of a special kind of data types, namely algebraic data types. Instances of these types are immutable values. The types can have structure, but only a hierarchical structure, namely composition. They represent sent messages, comparable to written and sent documents. Remaining in that metaphor, once you have sent a letter the content is unchangeable. In the data dictionary there must be a message type for each form chart state, and it must have the same name, except that the initial letter is lower case in the form chart, but upper case in the data dictionary.

The last diagram in the bundle that forms the specification is the semantic data model. This diagram is the conceptual data model that specifies the system state. Pure data record specifications which are needed by both, the semantic data model as well as the data dictionary, are put in a separate part of the data dictionary, the business signature repository. In our example the semantic data model is rather simple and consists mainly of the class holding the student information. The semantic data model is connected with the data dictionary again through the dialogue constraints, but also through so called opaque references.

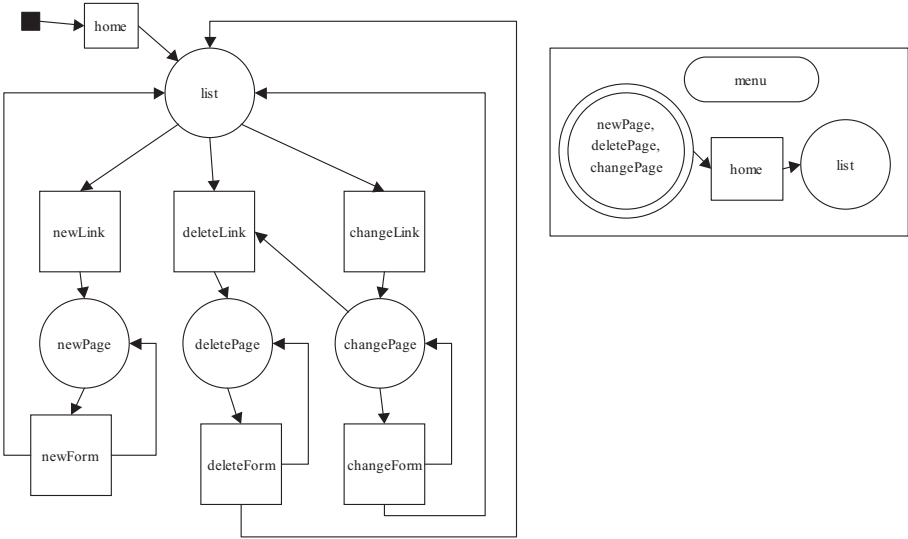


Fig. 2. Form chart of the seminar registration system.

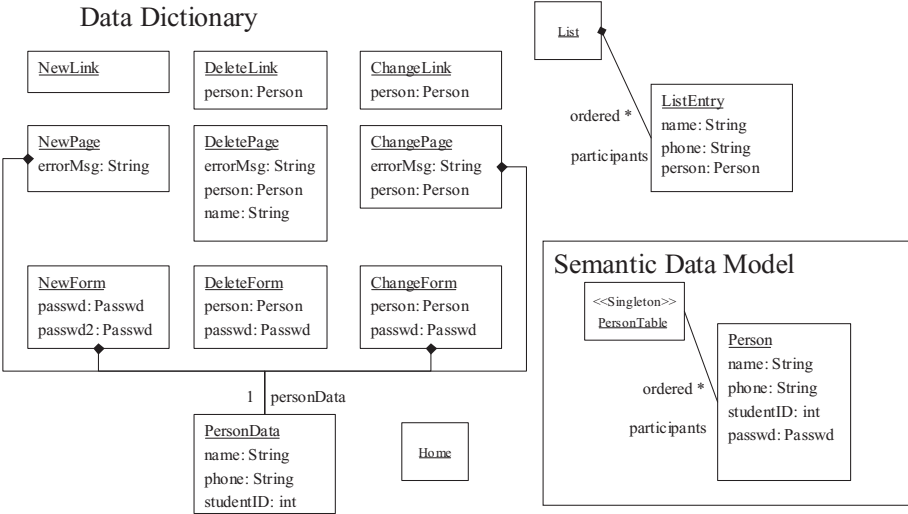


Fig. 3. Semantic data model and data dictionary

Take the client page `list` as an example. The message type `List` contains a list of different `ListEntry` objects that contain only the information presented to the user. This `ListEntry` submessage contains an attribute of type `Person`, the class from the semantic data model. Such a reference from the data dictionary to the semantic data model is called opaque reference. As one can see, if one follows the message types associated e.g. with the delete subdialogue, this reference is

passed along the dialogue and hence specifies, which person object is subject to the deletion process. The reference is passed along the form chart, yet the reference is opaque in the sense that only through certain operations that again access the semantic data model the content of the person object can be accessed.

The whole semantic data model forms a single data abstraction module with possibly as many opaque reference types as it contains classes. The opaque references are therefore the border of the data dictionary. The reference itself is part of the message, but not the referenced object. Therefore the object can change without violating our demand, that messages are unchangeable.

3 Dialogue Constraints

The message represents the signature of the state of same name. Each time this state is entered, a new message of this type has to be provided. We also specify signatures for the client pages. These client page signatures represent the information shown on the page. The page content is immutable. A page shows the same content to the user until she triggers a page change and therefore gets a new page, although possibly from the same type. Page interaction, i.e. user input in forms is not considered a change of the page content, but preparation of a new message. The fact that now the data dictionary contains the information shown on pages as well as the information sent back to the system as part of a page change is important with respect to the specification of so called dialogue constraints. Indeed one of the main advantages of form charts is that it allows elaborate constraint writing. We want to be able to express e.g. that the data record the user chooses for deletion must have been presented on the page. Such a constraint is called client output constraint. It is written in the following style.

```
list to deleteLink {
  clientOutput:
  source.participants.person->
    includes(target.person)
}
```

As we see in this example, we need the signature of the client page as well as the signature of the server action, called source and target, in order to set both in relation to each other. There are a number of different types of dialogue constraints, and they form together the dialogue constraint language, DCL for short. The DCL constraints are typically written in an attachment of the form chart, although in principle they can be written into the form chart diagram itself.

The Dialogue Constraint Language DCL introduces special purpose constraint types, which are shown in Figure 4. Transitions from client pages to server actions, page/server transitions for short, host two kinds of constraints, namely enabling conditions and client output constraints. An enabling condition

specifies under which circumstances this transition is enabled, based on the state during the last server action. The enabling condition may depend on the current dialogue history. The data submitted from a client page is constrained by the client output constraint. Server actions host server input constraints. They are server action preconditions in an incompletely specified system, they must be transformed to other conditions. Transitions from server actions to client pages, called server/page transitions for short, host flow conditions and server output constraints. The flow conditions specify for each outgoing transition, under which condition it is actually chosen. The server output constraint determines which information is presented on the client page that follows in the sequel. The client input constraint is a constraint on the information on the client page, which is independent from the server page.

The constraints in the form chart are written in a variant of OCL [18]. For this purpose OCL is enriched by new contexts and key labels with appropriate semantics due to the needs of dialogue constraint writing. Consequently data modeling is done with the pure data kernel of UML, whereby we distinguish message types in the so-called data dictionary from persistent data within the semantic data model. Persistent data can be accompanied by ephemeral session related data. The system functionality is seen as side effects of server actions. It may be specified in the context of the server action, but it typically will be structured by functional decomposition.

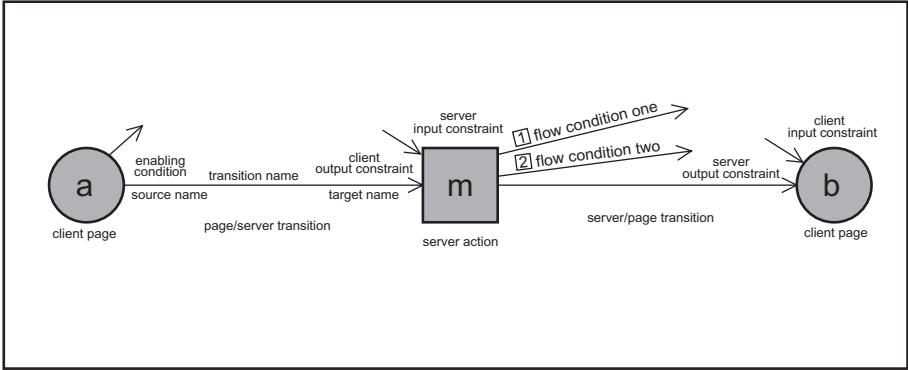


Fig. 4. Form chart notational elements

4 Feature Composition

Feature composition is introduced as the composition mechanism for form charts. The graph structure of a form chart has been specified as being a bipartite directed labeled multigraph. Every sub graph of the form chart is called a feature

chart. Two feature charts are combined by graph union. A form chart decomposition is a collection of feature charts in such a way that the combination of the feature charts yields the complete form chart.

The perhaps most intuitive explanation, why feature composition is possible and meaningful in Form-Oriented Analysis is the inverse operation, feature decomposition. A complete form chart has a uniquely stable semantics: If page/server edges, i.e. interaction options are removed, the data integrity is not endangered. Certain usages of the system may of course become impossible, if one removes key interaction options for the system. But the semantic data model is not corrupted by such operations: the system remains stable, if it was stable before. As a consequence the form chart covers system behavior that is inherently stable against runtime customizations.

The composition of the analysis model is of course especially important with respect to the task of expressing preferences and priorities in the system specification, as well as to enable the discussion of alternatives and trade-offs between them.

4.1 Compatibility Issues

There are some rules for the composition of two features. The rules follow from the fact that the features to merge must be subgraphs of one single form chart. First no node is at the same time client page in one graph and server action in the other. Nodes of the same name must have the same data dictionary type, because different features are different form charts over the same data dictionary and model.

If two features are combined, the constraints have to be compatible. If in a feature composition step a server action receives server/page transitions from different features, the flow condition numbers in both features must be different in order to be merged into a single order unless they are mutually exclusive. The server/page transition without flow condition has to be the same in both features, or one of the features should have no server/page transition without flow condition.

4.2 Hierarchical Feature Decomposition

A form chart can be decomposed in a hierarchical manner. The result is a tree of chart decompositions. Decomposition makes the form chart manageable. It is a tool for organizing the form chart artifact during the analysis phase. The feature hierarchy as such is not semantically relevant for the specification. Every combination of feature charts, even from different levels of the tree, yields a correct sub graph of the form chart.

4.3 Menu-Like User Interface Parts

An important special case of feature composition is the modeling of menu-like options, i.e. interaction options, which are offered on many, perhaps even all

pages. A new notation element for this purpose is the state set, that is depicted by a double lined state icon. It is annotated by a list of state names and serves as shorthand notation for these states. The example in Figure 5 shows page sets. An edge between two state sets of say m client pages and n server actions represents the complete bipartite graph $K_{m,n}$ between the elements of the state sets. A feature chart may be annotated as menu. Then the page/server transitions contained in this feature must not be contained in the main form chart or its decompositions. Affected states may reference the respective menu feature chart by an explicitly given name. Figure 5 shows how the described mechanism fosters readability of system interfaces with menu-like user interface parts. Another notation flavor is to give the state set a single name, and to reference the page set in its member states. The menu construct is used in the form chart of the seminar registration system in order to model the home button.

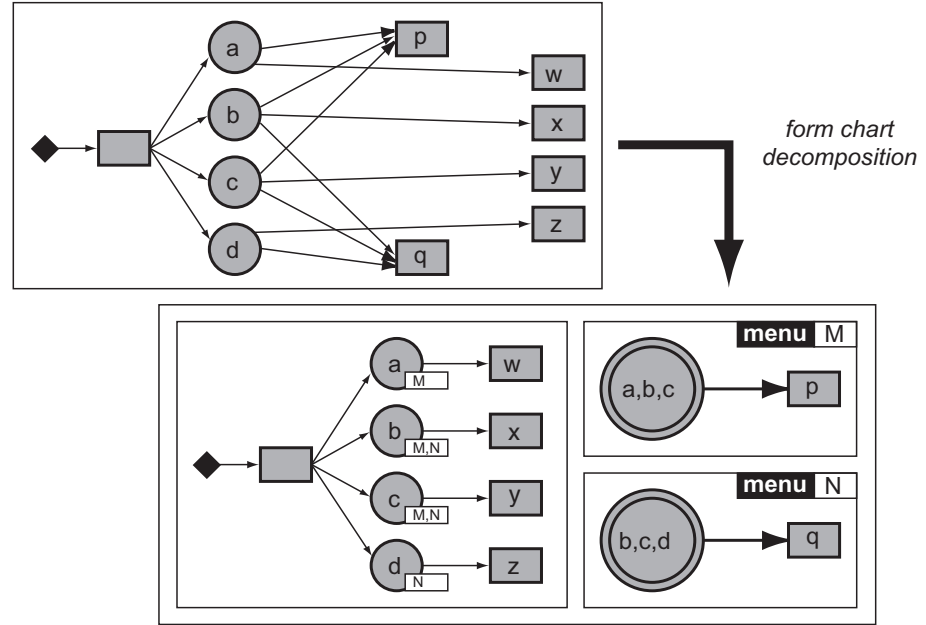


Fig. 5. Modeling menu-like user interface parts

5 Tool-Support for Form-Oriented Analysis

The model obtained in Form-Oriented Analysis can be transformed without impedance mismatch into an interface design and an implementation based on well-established technologies for web interfaces like server pages [6]. GENTLY is a proposed specification language for web-based presentation layers that provides

a textual format of form charts. The forward engineering tool GENTLY [8] and the design recovery tool JSPick [7] both exploit the specification language GENTLY. The GENTLY tool generates a complete prototypical dialogue based on Java Server Pages from a high-level system description in GENTLY. The JSPick tool generates high-level system descriptions for Java Server Pages based web presentation layers in a GENTLY dialect.

6 Related Work

Structured Analysis [15] is a very successful approach to both business modeling and system modeling that is still used in practice. It combines hierarchical data flow diagrams, sum-of-product data specification, local functionality specification and later [21] entity-relationship diagrams. The method is deliberately ambiguous with respect to the semantics of the several notational elements of the data flow diagrams and therefore heavily relies on the intuition of the modeler. Structured Analysis does not at all take into account driving forces of the solution domain.

The use-case driven approach to object oriented software engineering had deep impact. From the beginning [13] to state-of-the-art versions [14] of this approach the recommended human computer interface specification techniques exclusively target the modeling of GUI's. Furthermore the approach still lacks real world examples clarifying the meaning of use case specifications and how these can be exploited during system design and implementation.

State diagrams has been used for a long time in user interface specification [19], [10], [12], [20]), partly with the objective of user interface generation [4]. All of these approaches target user interface specification only at a fine-grained level, in our terminology concerning page interaction. Another early approach [11] targeted the modeling of push-based, form-based systems like the already disussed single-user desktop databases.

Within the UML Community the Discussion about dealing with the user interface is still underway [3]. In [5] a visual language for presenting user interfaces is proposed. The new artifacts are basically visualizations of page components. The method is tightly coupled with the use case driven approach. In our view, the diagrams do not reach the intuitive clarity of our proposed artifacts. A stereotype framework specifically for web applications is presented in [2]. This approach allows to model the design level concepts appearing during web site development with a typical web application framework. For this purpose the Conallan approach uses a set of stereotypes. The approach targets rather design than analysis.

Schwabe et al. presented a diagrammatic tool for representing web interaction [16], [17]. The diagrams are called user interaction diagrams (UID). They resemble page transition diagrams without server actions. Very restricted and very specific annotations are placed on the transitions concerning required selections by the user.

The aim to reduce the necessary navigation primitives is addressed in WebML [1], a visual language for conceptual modeling of complex web sites [1], in which all concepts are defined visually as well as in XML. WebML offers icons for page elements for composing web sites, e.g. catalogue pages and single item views. The WebML approach can be seen as an advanced and customizable successor of model driven interface generators.

7 Conclusion

Form-Oriented Analysis is an analysis technique for submit/response style applications. This class can be seen as a characterization of typical enterprise applications, including e.g. web applications. We model a submit/response style application with bipartite finite state machines, layered data models and dialogue constraints. Form charts are given by rigorous semantics and rules of usage. Our analysis technique is firmly based on existing well understood analysis notions and modeling techniques and consequently extends the state of the art in an important application domain: our analysis method is tailored to the class of submit/response style applications, but not designed as an analysis technique for all kinds of software systems. This strategic decision allows Form-Oriented Analysis to fit optimally to submit/response style applications and to provide added value for the analysis of such systems.

References

1. S. Ceri, P. Fraternali, and S. Paraboschi. Web Modeling Language(WebML): a modeling language for designing web sites. In *Proceedings of the 9 th. International World Wide Web Conference*, pages 137–157. Elsevier, 2000.
2. J. Conallen. Modeling Web Application Architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.
3. J. F. E. Cunha and N. J. Nunes. Towards a UML Profile for Interaction Design: The Wisdom Approach. In *Proc. UML'2000*, LNCS 1939. Springer, 2000.
4. P. P. da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems*, LNCS 1946, pages 207–226. Springer, June 2000. Limerick, Ireland.
5. P. P. da Silva and N. W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In *Proc. UML'2000*, LNCS 1939, 2000.
6. D. Draheim, E. Fehr, and G. Weber. Improving the Web Presentation Layer Architecture. In *In Proceedings of APWeb 2003 - The 5th Asia Pacific Web Conference*, LNCS. Springer, 2003. to appear.
7. D. Draheim, E. Fehr, and G. Weber. JSPick - A Server Pages Design Recovery Tool. In *CSMR 2003 - 7th European Conference on Software Maintenance and Reengineering*. IEEE Press, 2003. to appear.
8. D. Draheim and G. Weber. Specification and Generation of JSP Dialogues with Gently. In *Proceedings of NetObjectDays 2001*. transIT, September 2001. ISBN 3-00-008419-.

9. D. Draheim and G. Weber. Storyboarding Form-Based Interfaces. In *INTERACT 2003 - Ninth IFIP TC13 International Conference on Human-Computer Interaction*. IOS Press, 2003. to appear.
10. M. Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, 1987.
11. P. J. Hayes. Executable Interface Definitions Using Form-Based Interface Abstractions. *Advances in Human-Computer Interaction*, 1:161–189, 1985.
12. R. J. K. Jacob. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, 26(4):259–264, 1983.
13. I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.
14. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
15. D. Ross. Structured Analysis: A language for communicating ideas. *IEEE Transactions on Software Engineering*, 3(1), 1977.
16. P. Vilain, D. Schwabe, and C. S. de Souza. Modeling Interactions and Navigation in Web Applications. In *Proceedings of 7th International Workshop on Design, Specification and Verification of Interactive Systems*, LNCS 1921, pages 115–127. Springer, October 2000.
17. P. Vilain, D. Schwabe, and C. S. Souza. A Diagrammatic Tool for Representing User Interaction in UML. In *Proc. UML'2000*, LNCS 1939. Springer, 2000.
18. J. Warmer and A. G. Kleppe. *The Object Constraint Language*. Addison-Wesley, 1999.
19. A. I. Wasserman. A Specification Method for Interactive Information Systems. In *Proceedings SRS - Specification of Reliable Software*, IEEE Catalog No. 79 CHI1401-9C, pages 68–79. IEEE, 1979.
20. A. I. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transaction on Software Engineering*, SE-11(8):699–713, 1985.
21. E. Yourdon. *Modern Structured Analysis*. Yourdon Press, Prentice-Hall, 1989.

Semantic Web Services for Smart Devices in a “Global Understanding Environment”

Vagan Terziyan

Department of Mathematical Information Technology, University of Jyväskylä,
P.O. Box 35 (Agora), FIN-40014 Jyväskylä, Finland
vagan@it.jyu.fi

Abstract. Various Web resources and services are usually assumed to be used and accessed by human users (current Web) or by software agents on behalf of human users (emerging Semantic Web). However industry emerges also a new group of “users”, which are smart industrial devices, robots or any other objects, which can be adapted to the (Semantic) Web environment. They would need special services for e.g. online condition monitoring, information provisioning, remote diagnostics, maintenance support, etc. The goal of this paper is to specify main requirements to Web services that automatically follow up and predict the performance and maintenance needs of field devices. Semantic Web enabled services form a Service Network based on internal and external service platforms and OntoShell software. Concepts of a “Global Understanding Environment” and a “mobile service component” suppose that any component can be adapted to Semantic Web environment and executed at any platform from the Service Network, including service requestor side. This allows delivering not only a service results but also a service itself. Mobile service component within an OntoShell (agent) can move to a field device’s local environment (embedded agent platform) and perform its activities locally. Service components improve their performance through online learning and communication with other components. Heterogeneous service components’ discovery is based on semantic P2P search.

1 Introduction

The intersection of the following three domains have very recently started drawing enormous attention throughout academia and industry [3]:

- Web Service Technology (manifested through SOAP, WSDL and UDDI);
- Semantic Web Technology (manifested through ontology languages);
- Enterprise Integration (manifested through Enterprise Application Integration and E-Commerce in form of B2B Integration as well as B2C).

The promise is that Web Service Technology in conjunction with Semantic Web Technology (Semantic Web Services) will make Enterprise Integration dynamically possible for various enterprises compared to the “traditional” technologies, e.g. Electronic Data Interchange or Value Added Networks. Enterprise Integration will become more reliable as well as easier to achieve without the low-level implementation problems.

The Semantic Web is an initiative of the World Wide Web Consortium (W3C), with the goal of extending the current Web to facilitate Web automation, universally

accessible content, and the “Web of Trust”. Current trends on Web development leading to a more sophisticated architecture: Semantic Web; Device independence; Web Services. Tim Berners-Lee [2] has a vision of a semantic web, which has machine-understandable semantics of information, and trillions of specialized reasoning services that provide support in automated task achievement based on the accessible information. Web-enabled languages and technologies are being developed (e.g. RDF-Schema, DAML+OIL, OWL, DAML-S). The success of the Semantic Web will depend on a widespread adoption of these technologies.

Management of resources in Semantic Web is impossible without use of ontologies, which can be considered as high-level metadata about semantics of Web data and knowledge. Ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge [4]. DAML-S (DAML for Services [5]) provides an upper ontology for describing properties and capabilities of Web services in an unambiguous, computer interpretable markup language, which enables automation of service use by agents and reasoning about service properties and capabilities.

There is a growing interest in the use of ontologies in agent systems as a means to facilitate interoperability among diverse software components, in particular, where interoperability is achieved through the explicit modeling of the intended meaning of the concepts used in the interaction between diverse information sources, software components and/or service-providing software [12]. The problems arising from the creation, maintenance, use and sharing of such semantic descriptions are being highlighted by a number of recent large-scale initiatives supporting the interaction of heterogeneous systems (e.g. Agentcities, Grid computing, the Semantic Web and Web Services). A common trend across these initiatives is the growing need to support the synergy between ontology and agent technology.

Software applications can be accessed and executed via the Web based on the idea of interoperated Web services. A fundamental step toward this interoperation is the ability of automatically locating services on the bases of the functionalities that they provide. Location of web services is a semantic problem because the similarity and difference between services should be recognized on a semantic level. Current Web Services technology based on UDDI and WSDL does not make any use of semantic information and therefore fails to address the problem of matching between capabilities of services. In [13] the previous work on DAML-S was expanded, that describe service capabilities within DAML-S, can be mapped into UDDI records providing therefore a way to record semantic information within UDDI records. It was shown how to use this encoded information to perform semantic matching. Also efforts are made to improve DAML-S vision by inheriting some features from multi-agent systems community. For example in [7] it was mentioned that DAML-S approach does not separate the domain-neutral communicative intent of a message (considered in terms of speech acts) from its domain-specific content, unlike similar developments from the multi-agent systems community. To overcome this, an ontologically motivated Web Services system was presented the multi-agent systems techniques are discussed in the context of DAML Services model. The key component of the system is the central broker, which mediates the interaction between the other system components. Data consumers use the broker to find sources; the data sources register a service advertisement via broker; the broker responds with the matching services; the consumers then communicate directly with the data sources.

The key to Web Services is on-the-fly software composition through the use of loosely coupled, reusable software components [6]. Still, more work needs to be done before the Web service infrastructure can make this vision come true. Among most important European efforts in this area one can mention the SWWS (Semantic Web and Web Services, www.semanticweb.org) project, which is intended to provide a comprehensive Web Service description, discovery and mediation framework.

Usually a Web Service is expected to be accessed by human users or by software agents or applications on behalf of human users. However there already exists and growing a new group of Web Service “users”, which is smart industrial devices, robots or any other objects created by an industry and equipped by an “embedded intelligence” There is a good reason to launch special Web Services for such smart industrial devices. Such services will provide necessary online information provisioning for the smart devices, allow the heterogeneous devices to communicate and exchange data and knowledge with each other and even support co-operation between different devices. There are many questions to be answered within this research area. What is now the state of market for the smart devices and what are the trends and emerging needs of this market? Who should and who might be interested to launch appropriate Web Services? What might be a procedure, technologies, tools and resources to create such services? What kind of ontologies would be needed? In what extent Semantic Web – based approaches and technologies would be appropriate? How to manage data and knowledge of such services? How to manage multiple appropriate services? How to combine FIPA and W3C standards and apply it in Web Services to provide reasonable communication platform for smart devices?

In this paper we are trying to discuss the way of implementing emerging Semantic Web and Web services technologies to a real industrial domain, which is field device management. The goal of this paper is to specify main requirements to Web services that automatically follow up and predict the performance and maintenance needs of field devices.

The rest of the paper organized as follows. Chapter 2 describes the domain of field device management and maintenance and the concept of implementing agents in it in the framework of “Global Understanding Environment”. Chapter 3 presents the main requirements to the Web service network for smart devices based on integration of Semantic Web services’ and multiagent technologies. Main challenges of our approach are also summarized. Chapter 4 concludes.

2 GUN (Global Understanding eNvironment) Concept

The expectations from smart field devices include advanced diagnostics and predictive maintenance capabilities. The concerns in this area are to develop a diagnostics system that automatically follows up the performance and maintenance needs of field devices offering also easy access to this information. The emerging agent and communication technologies give new possibilities also in this field. Field device management in general consists of many areas of which the most important are:

- Selection
- Configuration

- Condition monitoring
- Maintenance

Valuable information is created during each phase of device management and it would be beneficial to save it into single database. This information can be utilized in many ways during the lifetime of the devices, especially as life cycle cost (or lifetime cost) of all assets is getting nowadays more and more attention. Accordingly the concept of life cycle management of assets has become very popular [14].

Field Agent is a software component that automatically follows the “health” of field devices. It is autonomous, it communicates with its environment and other Field Agents, and it is capable of learning new things and delivering new information to other Field Agents. It delivers reports and alarms to the user by means of existing and well-known technologies such as intranet and e-mail messages. Field device performance has a strong influence on process performance and reliable operation in more distributed process automation architecture based on FieldBus communication. In this situation, easy on-line access to the knowledge describing field device performance and maintenance needs is crucial. There is also growing need to provide automatic access to this knowledge not only to humans but also to other devices, applications, expert systems, agents etc., which can use this knowledge for different purposes of further device diagnostics and maintenance. Also the reuse of collected and shared knowledge is important for other field agents to manage maintenance in similar cases.

Among known cases to use agents in the maintenance domain, aircraft maintenance [17] supposes providing decision support for human mechanics. In an agent-supported process, a mechanic carries a wearable computer as he completes his maintenance tasks. When he encounters a discrepancy in his inspection, the mechanic fills out a form on his computer. The system analyzes the form and seeks out relevant information from agents. The system then displays the processed information for future use. The advantages of wearable computers with agents include automatic location and retrieval of information, utilization of historical repair data, and reduction in average time for repair.

We are considering case when (predictive) maintenance activities can be performed not only by humans but also by embedded automatics controlled by agents. We also assume that newest Semantic Web and Intelligent Web Services concepts can be applied to the problems of interoperability among field devices and will result to essential improvement of field device maintenance performance.

The concept of GUN (Global Understanding eNvironment) assumes an adaptation of every object from physical world to a Semantic Web environment. GUN Adapter is represented by integrated software/hardware components, which on the one hand implement object-specific functionalities and on the other hand – the common for whole Semantic Web environment functionalities. The Adapter translates interaction activities from device-specific format to a Semantic Web one and vice versa. Adapter also supplements real-world object with agent functionality, implicit purpose of the object becomes explicit goal of an agent (see Fig. 2).

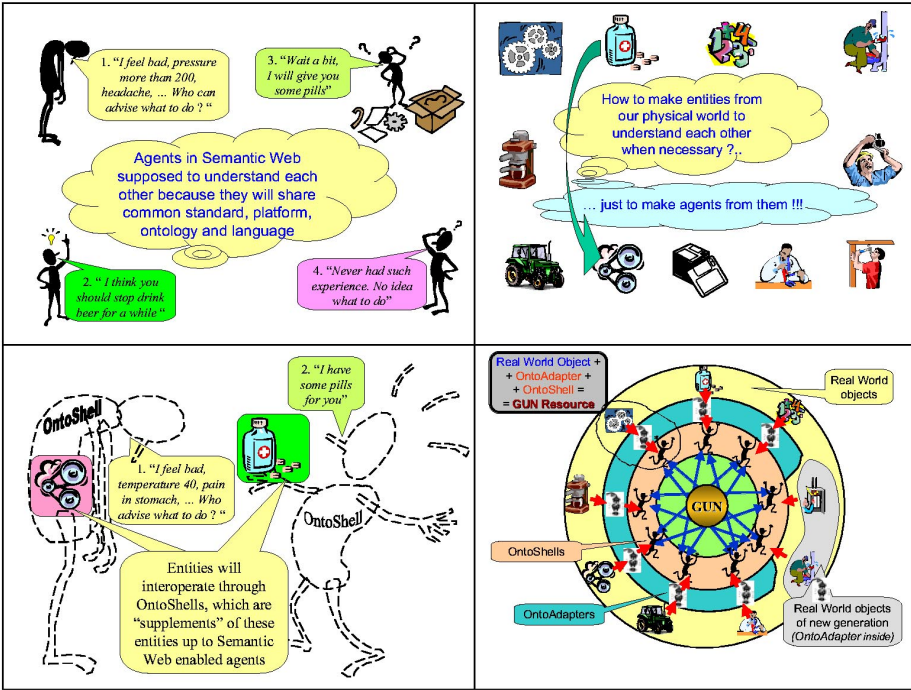


Fig. 1. The concept of Global Understanding eNvironment illustrated

The ideal GUN Adapter must adapt to a specific object automatically. The set of GUN agents can be joined into cluster (OntoShell) and the cluster will be represented for external world as a single entity. Example: industrial plant GUN agents (adaptive field devices) are joined into a cluster and other plants consider it as a single entity. As an example of implicit (“hidden”) purpose of an object we can take “pills”: they were manufactured for certain diseases and have strict application instructions. There are usually behind the scene producer and supplier of this product, some store, method, price and scope of delivery, business description etc. If to supplement the pills to the GUN agent and place it in some environment that supports such agents, then “owners” of the pills can forget about taking care of this object because an agent will take care about it. Most of present Web resources don’t have their purpose explicit: who can find it, what should be noticed, etc. An OntoShell is an active resource; an OntoAdapter supplements a passive resource with active functionalities. As a result a Semantic Web will be populated by active, goal-oriented agents. We consider a field agent concept and functionality related to field device monitoring and maintenance in the context of a GUN concept (Shells and Adapters).

Consider an example. Assume that the state of field device is described by 3 parameters and the dynamics of their change is it is shown in Fig. 2a.

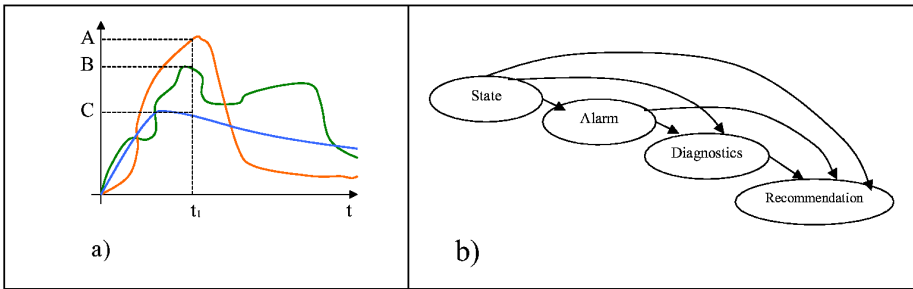


Fig. 2. The dynamics of data and its processing stages

Assume that in some time point t_1 we retrieved the values of the parameters: A, B, C. OntoAdapter integrated with the software, which provides these values, will wrap A, B and C with semantic metadata, basing on correspondent ontology of field device parameters. But what if there is some additional software that can determine the logical interval correspondent to the current value of field device parameter. The explicative example: the temperature of some field device varies in the interval from -10 C to $+100$ C. This interval is divided into 3 logical zones that can be recognized by alarm system: cool (-10 to $+10$), normal ($+10$ to $+60$), hot ($+60$ to $+100$). So, the outputs of alarm system can be also wrapped into semantic data based on additional ontology of intervals.

Fig. 2b shows possible data flows between software processing this data on 4 different levels. Alarm systems would need information about current state of field device. Diagnostic software can make decisions both on base of information about state and alarm condition.

Adapters can be really useful when many heterogeneous systems with different standards are integrated together. So there will be no need to change the components. However it is also true that sometimes it is more economically reasonable to implement a new Semantic Web compatible system by rebuilding the components instead of developing OntoAdapters for the old ones.

According to the GUN concept, every peer in Peer-to-Peer network of GUN resources is represented by correspondent OntoShell, which generalizes and hides its internal structure. Such peer-OntoShells are interconnected with each other as neighbours forming Peer-to-Peer network.

If to assume that each OntoShell accumulates knowledge about optimal query routes (routing information is semantically enriched), then one day a group of peers can make a decision about rearrangement. Such rearrangement means formation of peers cluster: peers are joined in a group according to some common features. For instance, peers, which provide different maintenance services for Control Valves can form a cluster 'Control Valve Services'. And this cluster generalizes the features of its members generating meta-profile for a group as for a single entity. Inside the group the nodes can be rearranged into centralized topology for more efficient query routing. The duties of a central (representative) node can be delegated to some of the internal peers. This peer will be an entry point to this cluster from the outside; it will possess the functionality of an OntoShell (see Fig. 3).

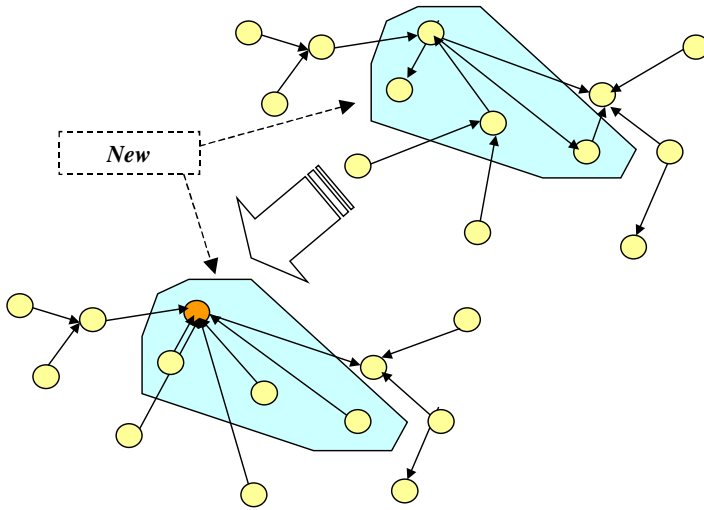


Fig. 3. Group of OntoShells is arranged in a cluster OntoShell

After formation of an OntoShell cluster a central peer can still have links with its former neighbours and even can be a member of another cluster. Such clustering of peers will reduce the unnecessary roams of queries. If the query doesn't match the profile of a cluster it wouldn't match any profile of its members. So the number of matching processes is decreased by number of members of the cluster. The challenge here is a process of generation of cluster meta-profile from the profiles of its members.

The concept of OntoShell can be also used for integration of formerly independent intranets. Each intranet has its own unique message protocol. OntoShell is “put” on each Intranet and every OntoShell translates messages from internal format into common for all OntoShells one. In this case a single node, which implements the functionality of OntoShell, must be created for every Intranet. This will be an entry point of external messages into the Intranet (see Fig. 4).

3 Network of Web Services for Smart Devices

Our primer goal is to implement the benefits of the Semantic Web (interoperability based on ontological support and semantic annotations), Intelligent Web Services (modeling, automated discovery and integration) and (Multi)Agent technologies (agents communication, coordination and mobility) according to GUN concept to substantially improve the performance of the Smart-Device Management Process by launching appropriate network of distributed intelligent maintenance services. More specifically the goal is to develop a an advanced global intelligent diagnostics system that automatically follows up and predicts the performance and maintenance needs of

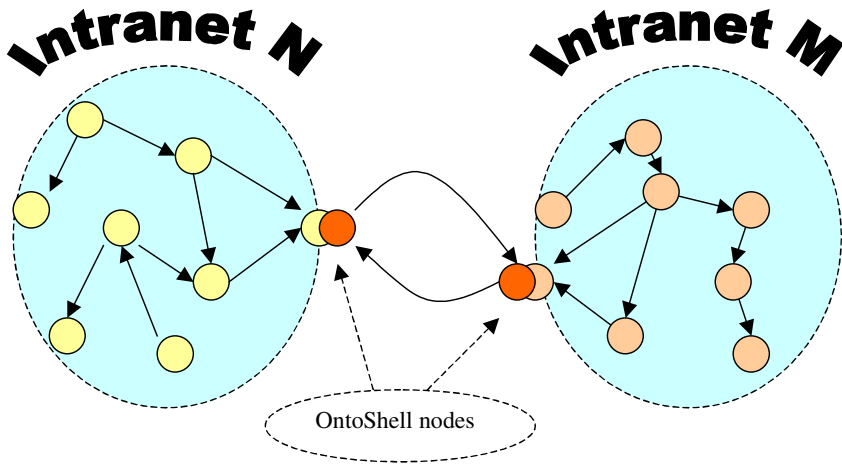


Fig. 4. OntoShells provide interoperability between Intranets

field devices, an appropriate multiagent architecture, ontological support for it, pilot implementation and case study.

3.1 Methods and Technologies Used

In this research we are using the most emerging technologies related to Semantic Web, Ontology Engineering, Semantic Annotations, Semantic Search, Intelligent Web Services (Modeling, Discovery and Integration), Multiagent Systems, Mobile Agents, Peer-to-Peer Networks, etc.; standards from W3C and FIPA, various Artificial Intelligence methods.

We base our efforts on intersection of Semantic Web, Web Services and (Multi)Agent Technologies, thus our approach inherits some essential properties of all of them:

- The services or service components are represented by agents and inherit following properties of *Agent Technology*: ability to learn, mobility, cooperation, coordination, FIPA standards and others. Network of services itself can be considered as multi-agent system of cooperating or self-interested, heterogeneous distributed agents;
- Inheritance from *Semantic Web Technology* results to the network of semantically annotated resources, which describe network of smart field devices that will be the subject of communication content of the agents.
- Profiles, roles and other properties of agents will be semantically annotated in a similar way as web-service profiles are annotated accordingly to *Web Services Framework*. To provide interoperability of main components within such framework we will need appropriate ontological support.

3.2 Main Requirements

Service requestors are smart-devices. The class of service requestors is extended with new group of service users – smart devices. We add semantic-enabled descriptions of services to facilitate automated search and use of services by smart-devices and to enable communication between heterogeneous services and agents acting on behalf of other services or human-user agents.

Services form a Service Network based on internal and external service platforms. Service Platform is an environment for running services. Services can be provided either locally, i.e. by embedding them to smart-device internal platform, or remotely by querying them from a Web-based external platform. Such external platforms form a distributed network of Web services, which is expanded by adding internal services platforms nodes. We assume that an external service can be queried either from Web-based external platform or from another internal platform. External Web service platforms provide more rich services since they are used by many clients and quality of services can be permanently improved according to growing experience. Various interactions between service platforms (internal-internal, internal-external, external-external) can be organized as a P2P-like network.

Service components are held by autonomous intelligent agents. Because of this we can benefit from agent technologies in a following way:

Service components are mobile. Our concept of “Mobile Service Component” supposes that any service component can be executed at any platform from our Service Network, including service requestor side. This allows delivering not only a service results but also a service itself when appropriate. Mobile service component representative (agent) can move to a local environment of service consumer (embedded agent-enabled environment, agent platform) and perform its activities locally. Necessity for such kind of services and their benefits can be shown on examples of services performing actions that demand close, intensive and secure communication with client-side platform. Here are several most typical cases:

- if there is more or less permanent need to use service component then the most appropriate solution is to import service component and use it on a client side;
- activities have to be performed locally for efficiency, business or security reasons;
- heavy data flows between client and service parts (intensive communication with service better be done locally rather than in remote fashion);
- necessity to provide guaranteed service availability if access to services network cannot be guaranteed for technical reasons (e.g. for wirelessly connected peers);
- strict services response time requirements (cross-network delay is too big, so local and fast service component is required).

Integration of mobile data components to create compound document was used in [15], where document is being dynamically composed of mobile agents and can migrate itself over a network as a whole, with all its embedded agents. The key of this framework is that it builds a hierarchical mobile agent system that enables multiple

mobile agents to be combined into a single mobile agent. In our approach we are also using mobile components but for creation of a compound service for smart devices.

Service components are able to learn. Service components can improve own performance through online learning during execution on service requestor site or through communication with other services components.

Service components are “Semantic Web Enabled”:

- Service components are semantically described;
- Service components are retrievable as a result of semantic search;
- Service components make decisions based on shared ontology;
- Service components can be integrated based on ontology.

Semantic P2P concept for service network management. The concept assumes decentralized management architectures with ontology-based information brokerage for:

- Service certification management;
- Service discovery management;
- Service responsibility management;
- Quality of Service management;
- Trust management
- Privacy and security management.

Existing security strategies are inadequate for the distributed networks, e.g. users in pervasive environments expect to access locally hosted resources and services anytime and anywhere leading to serious security risks and access control problems. In [8] a solution is proposed based on a *distributed trust management* infrastructure to provide a highly flexible mode of enforcing security in a pervasive computing environment. A Client can access the services provided by the nearest Service Manager via short-range communication. The Service Manager acts as an active proxy by executing services on behalf of any requestor.

The critical need to complement current navigational and information retrieval techniques with a strategy of information content and semantics is discussed in [10]. A scalable approach is proposed for vocabulary sharing. The objects in the repositories are represented as intentional descriptions by pre-existing ontologies characterizing information in different domains. User queries are rewritten by using interontology relationships to obtain semantics preserving translations across the ontologies.

The concept a Peer-to-Peer Semantic Web is discussed in [1]. Its realization assumes that DAML+OIL provides a specification framework for independently creating, maintaining, and interoperating ontologies while preserving their semantics, and P2P is used to provide a distributed architecture which can support sharing of independently created and maintained ontologies. The concept facilitates:

- Distributed and autonomous creation and maintenance of local ontologies,
- Advertisement (i.e., registry) of local ontologies,
- Controlled sharing of knowledge base components among users in the network,
- Ontology-driven semantic search of concepts and services,
- Knowledge discovery and exploration of inter-ontological relationships.

An emergent ad-hoc network of wirelessly connected agent platforms was investigated in [9]. Such network provides an environment for testing the multi-agent paradigm’s ability to provide interoperability between heterogeneous services. The multi-agent systems are moved into small physical and computational spaces, ensuring that they can discover and communicate with each other in a reliable and maintenance-free manner. Fraggile Rock (<http://sprocket.mle.ie>) is the smallest node deployed in the Agentcities network and remains interoperable with other FIPA-compliant platforms.

Certification of services on one hand, online learning of a trust network between intelligent distributed service components on the other hand, and finally P2P semantic discovery of service components in wired and wireless networks is the core of our approach to manage maintenance Web services and deal with security and privacy.

3.3 Extending Existing Maintenance Management Technology

The idea of having capabilities of browsing a device internal state extended to a browsing, automatic diagnostics and recovery within a network of maintenance centers is assumed to essentially extend the existing maintenance management technology.

The specifics of automation provided for smart-devices (that includes embedded condition monitoring) results to embedding in devices such components that can detect essential deviations from normal state of the device. For example a FieldBrowserTM [11] condition monitoring system from Metso Automation can detect faulty states and send error warnings (miscellaneous alarms) to e-mail addresses or mobile phones of operators located anywhere in the world. Another example is Sensodec 6S [16], which is the integrated solution to machinery condition monitoring and runnability analysis. The system analyzes the mechanical condition of machinery components. By using sophisticated analysis tools, machine operators and maintenance staff are quickly alerted to mechanical problems in their early phases of development.

Benefits of Condition Monitoring are evident from these examples. They are in:

- Avoidance of major mechanical failures, machinery damage and unplanned downtime;
- Immediate response is prompted by the operator interface;
- Effective preventive maintenance scheduling;
- Identification of exact causes of faults making maintenance quick and decisive.

However the problem still remains, which is lack of automatic diagnostic capabilities in existing solutions. Even such browsing solutions exist only for some types of devices and developed for a certain class of supported devices each. Current technology around UDDI, WSDL, and SOAP provide limited support in mechanizing service recognition, service configuration and combination, service comparison and automated negotiation.

Taking into account the above limitations we are using in our design a maintenance infrastructure with following main types of maintenance service components (agents):

1. *Wrapper components* – for integration with device-dependent (software and hardware) resources, acts as a semantic adaptor, mediator between semantic-enabled and traditional parts of service infrastructure;
2. *Management components* – for management of maintenance activities and distributed resource allocation;
3. *Diagnostic components* – for online discovery of problems within a device based on its state parameters and ontology-based classification of these problems (component is mobile agent);
4. *Recovery components* – for automatic planning and performing appropriate maintenance activities for a discovered diagnosis (component is mobile agent).

Humans can be represented in the maintenance services network as service component agents (human-service) or as a service requestor part (user agent), or both. These kinds of agents allow integration of humans into automated services network transparently for experts' decision support in diagnostic or recovery (maintenance) activities.

3.4 Ontological Support

The minimal set of necessary ontologies for the maintenance domain includes:

- Ontology of smart-devices;
- Ontology of maintenance activities;
- Device diagnostics ontology.

Also as it was mentioned above, there is also a need of ontologies for describing service components, trust, security, responsibility, quality, etc. profiles, in a way to allow implementation of the semantic P2P management framework

4 Conclusions

The goal of this paper is to specify main requirements to Web services that automatically follow up and predict the maintenance needs of field devices. Semantic Web enabled services form a Service Network based on internal and external service platforms. Concept of a "mobile service component" supposes that any component can be executed at any platform from the Service Network, including service requestor side. This allows delivering not only a service results but also a service itself. Mobile service component carrier (agent) can move to a field device's local environment (embedded agent platform) and perform its activities locally. Service components improve their performance through online learning and communication with other components. Heterogeneous service components' discovery is based on semantic P2P search. The paper contains mostly requirements and challenges related to Web services for smart devices in a Global Understanding Environment. More research and development efforts are needed to proof some of concepts mentioned in this paper.

Acknowledgements. Author is grateful to Dr. Jouni Pyotsia and his colleagues from Metso Corporation and Metso business units for useful consultations and materials.

Also I would like to thank my colleagues from Industrial Ontologies Group (Oleksandr Kononenko, Andriy Zharko and Oleksiy Khriyenko) for useful discussions within the scope of this paper.

References

1. Arumugam, M., Sheth, A., Arpinar, B.: The Peer-to-Peer Semantic Web: A Distributed Environment for Sharing Semantic Knowledge on the Web. In: Proceedings of International Workshop on Real World RDF and Semantic Web Applications, Hawaii (2002)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American*, Vol. 284, No. 5 (2001) 34–43
3. Bussler, C., Fensel, D., Sadeh, N.: Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, March 2003, available in:
4. <http://www.gvsu.edu/ssb/ijec/announcements/semantic.doc>
5. Chandrasekaran, B., Josephson, J., Benjamins, R.: What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, (January/February 1999) 20–26.
6. DAML Services Coalition: Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In: *The First International Semantic Web Conference (ISWC)* (2002)
7. Fensel, D., Bussler, C., Maedche, A.: A Conceptual Architecture of Semantic Web Enabled Web Services. *ACM Special Interest Group on Management of Data*, Vol. 31, No. 4 (2002)
8. Gibbins, N., Harris, S., Shadbolt, N.: Agent-based Semantic Web Services. In: *Proceedings Twelfth International World Wide Web Conference* (2003)
9. Kagal, L., Finin, T., Peng, Y.: A Framework for Distributed Trust Management. In: *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control* (2001)
10. Lawrence, J., Embedded FIPA Agents. In: *Agentcities: Agent Technology Exhibition*, Barcelona (2003), Available in: http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf
11. Mena, E., Illarramendi, A., Kashyap, V., Sheth, A.: OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation Across Pre-existing Ontologies. *International Journal on Distributed and Parallel Databases*, Vol. 8, No. 2, (2000) 223–271
12. Neles FieldBrowser™ System for Field Device Predictive Maintenance. *Metso Automation Techn. Bulletin* (2003) Available in: <http://www.metsoautomation.com/>
13. Ontologies in Agent Systems: Workshop announcement (11 March 2003) Available in: <http://oas.otago.ac.nz/OAS2003>
14. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Importing the Semantic Web in UDDI. In: *Proceedings of Web Services, E-business and Semantic Web Workshop* (2002)
15. Pyötsiä, J., Cederlöf, H.: Advanced Diagnostic Concept Using Intelligent Field Agents. *ISA Proceedings* (1999)
16. Satoh, I.: Mobile Agent-Based Compound Documents. In: *Proceedings of the 2001 ACM Symposium on Document Engineering*, ACM Press (2001) 76–84
17. Sensodec 6C for Paper: Metso Automation Technical Bulletin (2003) Available in: <http://www.metsoautomation.com/>
18. Shehory, O., Sukthankar, G., Sycara, K.: Agent Aided Aircraft Maintenance. In: *Proceedings of Autonomous Agents '99*, Seattle, Washington (1999) 306–312

Construction of Generic Web-Based User Interfaces

Arno Puder

San Francisco State University
1600 Holloway Avenue
San Francisco, CA 94132
arno@sfsu.edu

Abstract. Several middleware technologies exist today that facilitate the development of applications in distributed heterogeneous environments. Interoperability is the property by which an application can span different middleware technologies. Much attention has been given to the application-to-application interoperability. With the rise of web services, we introduce a generic model for user-to-application interoperability. At the core of this concept is a mapping from user-friendly web forms to arbitrary middleware technologies.

1 Motivation

A middleware seeks to facilitate the development of applications in distributed, heterogeneous environments. Several predominant middleware technologies exist today, such as CORBA, EJB or Web Services. As applications often span different technological and administrative domains, it is not uncommon that one application is deployed on several different middleware platforms.

Interoperability defines the property by which different middleware technologies connect to each other. Without interoperability, an application could not span different middleware technologies. Much attention has been paid to interoperability for the predominant middleware technologies. Today there exist solutions to connect the major middleware platforms.

We view the technologies around web applications as another middleware platform. The web browser serves as a generic user interface. Applications are written as Common Gateway Interface (CGI) scripts that run on the side of the web server and communicate with the web browser via HTML over HTTP. Several technologies exist that facilitate the development of web applications such as Java Server Pages (JSP) or Active Server Pages (ASP). While highly flexible, programs written in this technology often merely serve as a bridge between the web browser and some backend service such as a middleware or a database.

In this paper we introduce a model that allows the automatic creation of web-based user interfaces and that does not require any coding on the side of the web server. We describe a generic bridge that can be used to create user-friendly interfaces for arbitrary backend services without having to implement

a custom interface. The interface is derived automatically from the interface specification of the backend service.

In section 2 we take a closer look at interoperability in the context of user-to-application interoperability. Section 3 introduces the architecture of our generic bridge. Some implementation details are highlighted in section 4 while section 5 discusses related work. Finally, section 6 provides a conclusion and outlook.

2 Interoperability

Development of distributed applications is supported by a middleware. The term “middleware” derives from the fact that it is located between the operating system and the application and provides a level of abstraction for distributed applications. One way to think of it is that a middleware platform spreads out like a table-cloth in a heterogeneous environment, hiding different technologies beneath it. The same API is offered at the various access points throughout the network.

Unfortunately it is impossible to impose the same technology everywhere and it can not be avoided that different middleware technologies dominate in different domains. The dashed line in Figure 1 defines the interface between two different technological domains. By standardizing this interface, one achieves *interoperability* of applications thereby decoupling technological domains.

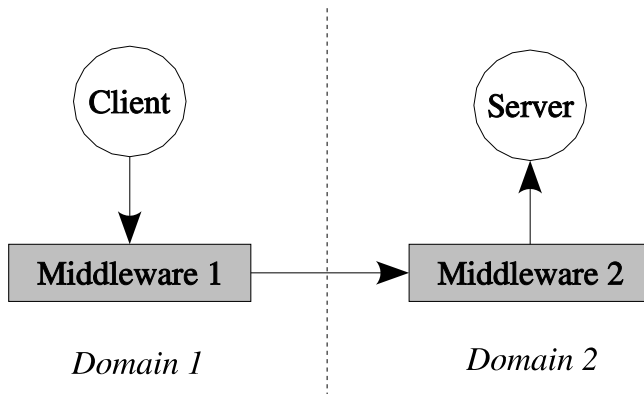


Fig. 1. Interoperability between different domains.

The interface between different middleware technologies is usually defined through a network protocol detailing how the two platforms talk to each other. Interoperability of two domains can only be achieved if the object invocation semantics and type systems that are used to describe object interfaces can be mapped onto each other. Sometimes this leads to an extension of the type system of one technology. E.g., in order to achieve interoperability between CORBA and

EJB, the CORBA type system had to be augmented by value-type-semantics (a.k.a. objects-by-value). Another way of overcoming differences is to introduce a bridge that implements a custom mapping.

Table 1. Comparison of different middleware technologies

Middleware	Interoperability
DCOM	Proprietary, DCOM specific protocol.
EJB	Uses Java specific RMI or optional CORBA-IIOP for the transport layer
SOAP	XML-based marshalling
CORBA	Defined through GIOP/IIOP
Web	HTML over HTTP/CGI

Table 1 summarizes the characteristics of various middleware technologies in use today with respect to the way they define interoperability. We view the infrastructure for web applications as another middleware technology. As applications get increasingly end-to-end, reaching from the user interface to some backend system, we focus our attention to interoperability with web applications.

Web applications are typically implemented as Common Gateway Interface (CGI) scripts that get invoked by the web server. These scripts process any user input and act as a bridge to some backend service. Common techniques for implementing these CGI scripts are Java Server Pages (JSP) or Active Server Pages (ASP) that embed a programming language inside an HTML-page.

For many applications, the CGI scripts merely act as a bridge to a backend service. Data validation, implementation business logic and database access are typically implemented on a backend system and the CGI script only passes user input to this backend and renders an HTML page as a result.

In this paper we introduce a generic bridge that is implemented as a CGI script. The bridge automatically creates user-friendly user interfaces based on the operational interface of the backend system. This approach is particularly feasible for course-grained, loosely coupled systems, because there is a natural mapping between the operational interface and a user interface. This is particularly useful for web services which connect loosely coupled systems.

The advantage of this approach is that no programming is required to build a user interface. This makes it easy to deploy new services quickly and allow access through a regular web browser. The downside of our approach is that the look-and-feel of the user interface is determined by the generic bridge and sometimes it still is preferable to build a custom user interface.

3 Architecture

Figure 2 shows the overall architecture of the generic bridge. The bridge runs as a CGI-script at the side of the web server. A general purpose connector allows

it to access different backend technologies, such as SMTP, SQL, CORBA, and Web Services. As will be shown later, the generic bridge can be given “hints” expressed in XML on how to render the user interface. The following sections give a detailed overview of the various components of the generic bridge.

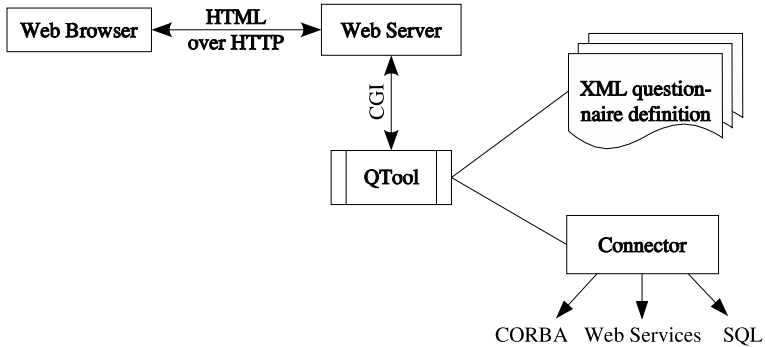


Fig. 2. Architecture.

3.1 User Interface Model

This section introduces a general user interface model. The model defines the basic building blocks that can be used to define a user interface. While there will certainly be special cases where this user interface model is not powerful enough, we believe that it can be used in a variety of different scenarios. A later section will give an overview where the generic bridge has been successfully used.

The heart of the user interface model is the notion of a *questionnaire*. Abstractly, a questionnaire is a user-friendly visualization of an operational interface. A questionnaire is composed of one or more *sections* that allow the logical grouping of questions. Sections can be nested (a section within a section) and can furthermore be marked as repeatable. For repeatable sections, the user can input more than one instances of this section.

Each section contains one or more *questions*. A question prompts the user for one specific feature. It is rendered in the web browser as an input field. Each question has an associated *question type* that determines the kind of response that is expected from the user. Examples of question types are string, textfields, external documents, etc. Table 2 gives a list of all the question types supported by the generic bridge. Each question type is rendered as a specific HTML element such as a drop-down list or a checkbox.

All components of the general user interface model (questionnaire, section, question) can furthermore contain documentation that will be included in the rendering of the questionnaire. In summary, the general user interface consists of the following components:

Table 2. Question types

Type	Description
String	One line text response
Textfield	Several lines text response
Email	Email address
URL	Web-URL
Bool	Yes/No response
Checkbox	Yes/No response as a checkbox
Feature	Possible values: Yes/No/Unknown/Planned
Ranking	Possible values: 1–10, N/A
Document	External document

Questionnaire:

- Documentation
- List of Sections

Section:

- Documentation
- Can be marked as repeatable
- Contains one or more questions
- Sections can be nested

Question:

- Documentation
- Can be marked as mandatory
- Has a type (see Table 2)
- Question text

The questionnaire is defined through an XML document. Each of the components of the questionnaire (sections and questions) are represented by appropriate XML-tags. In that sense, the generic bridge defines a new XML-application by providing the “vocabulary” to express the questionnaire. The following shows an excerpt of a question definition:

```
<question name="DII" type="feature" mandatory="no">
  <doc>
    The Dynamic Invocation Interface (DII) is part of the
    client side API. With the help of the DII, a client can
    construct method invocation at runtime without the need
    for an IDL-generated stub.
  </doc>
  <query>
    Dynamic Invocation Interface (DII)
  </query>
</question>
```

The question above was taken out of a questionnaire for CORBA products. This particular question asks about the availability of the Dynamic Invocation

Interface (DII) of a given CORBA product. The question type “feature” has the following four possible values: yes (DII is supported), no (DII is not supported), planned (vendor plans to implement the DII) and unknown (it is unknown whether the DII is supported). The question type “feature” is useful for capturing the features of a product.

Based on the specification of the question a HTML-based user interface is automatically generated (see screenshot depicted in Figure 3). The documentation of the question is displayed in a popup window when clicking on the hyperlink of the question. The question type feature is rendered as a selection box (the user has selected “Planned” in the screenshot). The other elements of the screenshot displayed in Figure 3 such as the title “CORBA” or the section title “CORBA Core” are also defined in the questionnaire and are described by appropriate XML tags.

Fig. 3. HTML-based interface for the DII question.

Since a questionnaire is based on an XML-document, it is possible to describe its syntax through an XML schema. This schema is itself an XML document, but it describes the legal instances of a questionnaire. The following excerpt shows the XML schema definition of a question. Note that the above example of the DII question complies with the syntax defined here:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

[... ]

<xsd:complexType name="QuestionType">
  <xsd:sequence>
    <xsd:element name="doc" type="xsd:string"/>
    <xsd:element name="query" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

```

<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="type" type="xsd:string"/>
<xsd:attribute name="mandatory" use="optional" type="YesNoType"/>
</xsd:complexType>

```

```
[...]
```

```
</xsd:schema>
```

The above definition expresses that a question consists of a documentation field and a query field. A question must have a unique name and type which are specified as XML attributes. Furthermore, a question can be marked as mandatory or optional. With the former the user is required to provide input for that particular question.

3.2 Meta-questionnaire

In the previous section it was said that the schema of a questionnaire has to be defined through an XML document. This XML document contains all the information that are needed by the generic bridge to render a user-interface. While this approach is very flexible, it requires the end user to provide an XML document for each new questionnaire.

Since the emphasis of the generic bridge is end-user friendliness, it is not acceptable to expect knowledge of XML. For this reason, a web-based management interface is provided that allows the definition and maintainance of questionnaires.

To define a new questionnaire, the user has to fill out a special questionnaire whose purpose is to ask for all the elements of the new questionnaire to be defined. One important insight is that this “meta-questionnaire” is yet another questionnaire. I.e., in order to define a new questionnaire, the user has to fill out a special questionnaire. The special, or meta-questionnaire, contains all the questions that need to be answered in order to define a new questionnaire.

The screenshot depicted in Figure 4 shows the rendered HTML interface of the meta-questionnaire for the DII question introduced in the previous section. First notice that the look-and-feel of the user interface is similar to that of the DII question depicted in Figure 3. The various questions displayed in the screenshot mirror the information that was specified in the XML document of the previous section. This shows that new questionnaires can be defined using the meta-questionnaire and without any knowledge of XML.

Using a meta-questionnaire has two main benefits. The first is that the user is alleviated from having to have any knowledge of XML to specify new questionnaires. Instead the user can use the familiar user interface that queries everything the generic bridge needs to know about the new questionnaire. The second benefit it that the implementation of the bridge is greatly simplified. Instead of having a separate management interface, questionnaires are managed through the familiar user interface itself. The same code can be re-used.

2.2. Question

Question text: (*)
Question type:
Question mandatory:
Question documentation:

Dynamic Invocation Interface (DII) is part of the client side API. With the help of the DII, a client can construct method invocation at runtime without the need for an IDL-generated stub.

Fig. 4. Meta-questionnaire.

3.3 Connectors

So far we have discussed two different ways of defining a questionnaire: through a XML document and via the meta-questionnaire. Another way to create questionnaires is to derive them from operational interfaces such as CORBA–IDL or WSDL (Web Service Definition Language). A connector translates between questionnaires and such operational interfaces. The translation requires mapping rules. This mapping is bidirectional: questionnaires can be mapped to operational interfaces and vice versa. In the following we focus on the former. I.e., we assume the existence of a questionnaire and mapping rules map this questionnaire to a backend interface. The reverse mapping is subject to future work.

All the input that is collected by one questionnaire is called a *profile*. Once the generic bridge receives a profile, it has to be forwarded to some backend system. Thus the profile becomes the actual parameter for an invocation of the backend system. The connector represents an abstract interface to various backend technologies (see Figure 5). Through inheritance, special purpose adapters can be created. For each of the special purpose adapters one needs to define a mapping between questionnaires to data types of the specific technology. In the following we describe the mapping for various connectors.

Mail. The mail connector collects all user responses to a questionnaire, marks them up as an XML document and sends them to a designated email address. This configuration is useful for user feedback via email. Unless the data is not

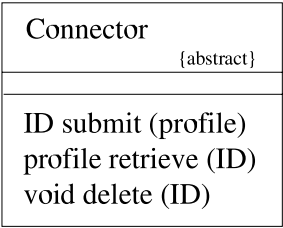


Fig. 5. UML diagram for the connector.

passed to another connector, the profile is simply sent as an email. The mail connector by itself does not store the profile. Right now the content of this email is an XML document. It is possible to add XSLT support to create a more human readable mail.

Database. In many cases it is desirable to store profiles in a database. The profile becomes persistent and can be viewed and edited at a later point in time. For this reason our generic bridge includes a connector for relational databases.

Unlike the mail connector described in the previous subsection, the database connector requires a mapping of a questionnaire to a relational database schema. This mapping is defined once and then implemented within the database connector, so that the schema for the database can be automatically created out of the questionnaire definition.

Mapping of a questionnaire to a relational database schema is straightforward. Each question in the questionnaire is mapped to a column of a table representing the questionnaire (see Appendix A). Retaining the logical grouping introduced by sections is not necessary, so that the structure is flattened into a set of questions.

Repeatable sections pose a special challenge since a profile can contain multiple instances of this section. It is not known a priori how many instances a user might supply. For each instance the schema of the table would need to be extended by adding an appropriate number of columns. One way to map repeatable sections is to introduce a second table whose columns represent the questions in that repeatable section. Multiple instances of this section are represented by different rows of that table. Those rows that belong to one profile are linked via a foreign key of the relational database with the table representing the questionnaire itself.

CORBA. A CORBA connector acts as a bridge between a user and a CORBA backend. Profiles submitted by the user are forwarded via CORBA object invocations to a backend server. Since in CORBA the interface of an object is specified via the Interface Definition Language (IDL), a questionnaire must be mapped to an IDL-specification. The connector is then using the Dynamic Invocation Interface (DII) to translate profiles to object invocations.

A questionnaire is mapped to a CORBA interface. Each question type is mapped to an appropriate CORBA type. Sections are mapped to an IDL-structure. The nesting of structures mirrors the nesting of sections. In the case of CORBA, repeatable questions pose less of a problem than with the database connector discussed in the previous section. A repeatable section is mapped to an IDL-sequence that can have zero or more instances of that section.

Web Services. The mapping for web services is similar to the one described for CORBA in the previous section. In web services, the interface of an object is defined through the Web Service Definition Language (WSDL). Just as for CORBA, a mapping needs to be defined to map a questionnaire to a WSDL specification. The web service connector generates SOAP messages based on the schema of a questionnaire.

The web services mapping follows the same pattern as the mapping for CORBA. Question types are mapped to appropriate WSDL types. Sections are mapped to structures. Repeatable sections are mapped to sequences of structures.

4 Implementation

The concepts described in this paper have been implemented and have become known under the name QTool (for questionnaire tool). QTool is implemented using the scripting language Python (see [5]). Python has proven to be very suited for this kind of application. In our opinion it is easier to learn and maintain than Perl. In this section we describe some of the implementation details.

4.1 Commands

QTool runs as a CGI script invoked by the web server. Both Apache and Microsofts IIS are supported. The functionality of QTool can be accessed through the usual GET and POST conventions of the HTTP protocol. E.g., to get an empty questionnaire for a schema called “CORBA”, the following URL need to be invoked:

```
http://<host>/cgi-bin/qtool.cgi?ACTION=NEW&PROFILE=CORBA
```

There are several different commands with different arguments that can be accessed in a similar way. These URLs can be used as hyperlinks of anchors within a custom HTML page. It is also possible to access the functionality of QTool through Server Side Includes (SSI). With this technology a specially marked up HTML page contains inline calls to a CGI script. The web server will replace those inline invocations with the output generated by the CGI script. This allows custom HTML pages that embeds the output of QTool in a very flexible way.

4.2 PSP

QTool is implemented in Python and is invoked as a CGI script from a web server. The script's main task is to generate HTML that is rendered in the client's browser. There are different approaches to generating the HTML, one popular one being *Java Server Pages* (JSP). In JSP, the web page is interspersed with Java code that emits the dynamic part of that page. Several projects have adopted this scheme for Python, which consequently is called *Python Server Pages* (PSP) (see [1]).

One challenge of PSP in contrast to JSP is the fact that in the language Python programming blocks are marked through different indentation levels. Therefore, instead of marking a block with '{' and '}' as done in Java, the statements belonging to a block in Python have the same indentation level.

This poses a challenge when applying the JSP idea to Python, because it is difficult to maintain the indentation level when HTML code is interspersed in a Python program. One solution to this problem is to extend the Python language by adding explicit begin and end markers for programming blocks, as done in X.

For QTool we have taken a different approach. While JSP is based on the principle "HTML code with interspersed Java code," our version of PSP reverses this principle to "Python code interspersed with HTML code." The Python code defines the level of indentation and the HTML part is embedded naturally at the respective indentation level. The following example illustrates this:

```
# PSP code
for i in range (10):
    <!
    Hello World.
    <P>
    !>
<!
The end.
!>
```

An HTML block is enclosed with the markers "<!" and "!>". Inside this block all code is treated as HTML and not as Python code. Note that the embedded HTML code follows the indentation rules of Python. The PSP code above will generate ten "Hello World" followed by one "The end." In the generated HTML code, the whitespaces that are required to mark the level of indentation within the PSP program will be removed.

Similar to JSP, our adaptation of PSP allows the embedding of Python expressions within an HTML block. The following code excerpt illustrates this:

```
# PSP
numbers = ['one', 'two', 'three', 'four', 'five']
<!
Counting from 1 to 5:
!>
for i in numbers:
    <!
    Number <% i %>.
    <br>
!>
```

4.3 Connectors

One of the central concepts in QTool is that of a connector that serves as an abstract interface to various backend technologies. Since QTool is written in Python, specific connectors need to be implemented as derived classes in Python. The mail connector uses Python's builtin SMTP library. The CORBA connector uses Fnorb; a Python-based CORBA implementation (see [7]). For web services, QTool makes use of SOAPpy, a web service implementation for Python (see [8]). Finally, the relational database connector makes use of a Python interface to MySQL (see [3]).

4.4 Examples

The concepts presented in this paper have been successfully implemented. The resulting implementation is placed under the GPL license and can be downloaded from <http://www.puder.org/qtool/>. QTool has been used for various projects. One of them is the San Francisco Movie page that lists movies made in that city. Besides the movies, this page also shows locations where certain scenes of a movie have been made. This feature makes use of repeatable sections, where a section called "Location" is marked as repeatable.

Another project where QTool has been used is the CORBA Product Matrix. This page gives a visual overview of the features of over two dozen CORBA products. Both the movie page and the CORBA product matrix use the database connector to store and maintain the information submitted by the user. The QTool homepage mentioned in the previous section contains links to those two QTool applications.

QTool has also been used within AT&T Labs for product evaluations. The relevant criteria of the product under evaluations are collected in a questionnaire for further analysis. A team of experts can easily submit their feedback by answering the evaluation questionnaire. In a different project within AT&T Labs, QTool is used as a generic front-end for a workflow engine. This particular application uses the web services connector to talk to the workflow engine.

5 Related Work

We take a pragmatic approach to define the general user interface model. This model is inspired by the recurring design pattern of operational interfaces. This facilitates the mapping between operational and user interfaces. Question types have a natural mapping to types of various backend technologies. Repeatable and non-repeatable section can be mapped to constructed data types.

Web service technologies often include test interfaces where a primitive web-based user interface is automatically generated out of a WSDL (Web Service Definition Language). These are generally only meant for testing purposes and do not support complex data types. Examples are .NET, WebLogic Server, or XML Gateway. In previous work we have demonstrated that a knowledge representation technique called Conceptual Graphs can be used as a generic interface to CORBA's DII (see [6]). While very flexible, this approach is not suitable for end users.

Other approaches for general user interfaces focus on the data modelling aspect and usually have some cognitive model of the real world at the core of their design. Examples are the embodied construction grammars (see [2]) or the formal concept analysis (see [4]). While these approaches offer comprehensive solutions to represent arbitrary domain knowledge, they are too heavy-weight and not user friendly for general purpose user interfaces.

6 Conclusion and Outlook

Interoperability is one of the key issues when developing distributed applications. Interoperability is an end-to-end issue that spans from the user front-end to the backend. One prominent way to implement user front-ends is via a web-browser. In this paper we have introduced a way to dynamically create user interfaces for different backend technologies such as relational databases and different middleware technologies.

One future extension will revolve around the layout of the user interface. Currently, the layout is hard-coded in the implementation itself. We plan to use XSLT to change the look-and-feel through XSLT-style sheets. As another extension we intend to take a closer look at WSDL, the interface definition language for web services. Since both QTool questionnaires and WSDL are expressed through XML, it might be an interesting idea to merge these two formats.

References

1. R. Barr. Python Server Pages. <http://spyce.sourceforge.net/>, SourceForge, Cornell University, 2002.
2. B. Bergen and N. Chang. Embodied construction grammar in simulation-based language understanding. In Jan Ola Ostman and Mirjam Fried, editors, *Construction Grammars: Cognitive and Cross-Language Dimensions*. Johns Benjamins, 2002.

3. A. Dustman. MySQL for Python. <http://sourceforge.net/projects/mysql-python/>, SourceForge, 2003.
4. F. Lehmann and R. Wille. A Triadic Approach to Formal Concept Analysis. In *3rd International Conference on Conceptual Structures (ICCS'95)*, Santa Cruz, University of California, 14–18 August 1995. Springer Verlag.
5. M. Lutz. *Programming Python*. O'Reilly & Associates, second edition, 2001.
6. A. Puder and K. Römer. Use of Meta-Information in a CORBA Environment. In *Workshop on CORBA: Implementation, Use and Evaluation*, Jyväskylä, Finland, 1997. European Conference on Object-Oriented Programming (ECOOP).
7. R. Smith. Fnorb: A Python-based CORBA implementation. <http://sourceforge.net/projects/fnorb/>, SourceForge, Distributed Systems Technology Centre (DSTC), 2002.
8. C. Ullman and B. Matthews. SOAPpy: Web Services for Python. <http://sourceforge.net/projects/pywebsvcs/>, SourceForge, 2003.

A Mapping

The table below summarizes the mapping of the various question types to SQL, CORBA, and Web Services types. Note that the mapping is bidirectional. All question types have a corresponding mapping for the various backend technologies discussed here. The SQL mapping makes use of builtin types and tables. The CORBA mapping uses IDL-types to map the various question types. The web service mapping makes use of types defined in XML schema. Currently some types (e.g., a CORBA-`struct`) can not be mapped to a questionnaire. While a mapping is possible, this will be subject to future work.

Type	SQL	CORBA	WS
String	TINYTEXT	string	<xsd:string>
Textfield	MEDIUMTEXT	string	<xsd:string>
Email	TINYTEXT	string	<xsd:string>
URL	TINYTEXT	string	<xsd:string>
Bool	ENUM	boolean	<xsd:boolean>
Checkbox	ENUM	boolean	<xsd:boolean>
Feature	ENUM	enum	<xsd:enum>
Ranking	ENUM	enum	<xsd:enum>
Document	LOB	sequence<octet>	<xsd:sequence>
Questionnaire	TABLE	interface	<service>
Non-repeatable section	TABLE	struct	<xsd:sequence>
Repeatable section	TABLE (foreign key)	sequence<struct>	<xsd:sequence>

A Word Sense Disambiguation Algorithm for Information Retrieval Applications

Giuliano Pascucci and Sandro Spadaro

Neotek Solution s.r.l., via V. Giuffrida, 121 – 95100, Catania
giuliano@aconet.it, s.spadaro@neoteksolution.it

Abstract. After a brief introduction and overview of some remarkable studies about Natural Language Processing and Word Sense Disambiguation, the authors describe a general purpose statistic method for the disambiguation of polysemous words in English.

Unlike other methods, the one here introduced needs no linguistic or grammatical restrictions to produce effective results.

Such a method consists in an algorithm based on the co-occurrence and frequency of words in the Internet, and avails itself of WordNet 1.6 and Altavista Search Engine.

The results of this study are also presented and discussed.

In the final section of the paper, possible future applications for the devised algorithm are described.

Introduction

The birth of the World Wide Web (WWW) (Berners-Lee 94) and related internetworking technologies has been accompanied by an explosion of the types, availability, and volume of data accessible to an information system.

The overload of information scattered in various repositories in the World Wide Web has increased the necessity for a semantic web, since at present, the data on the WWW are not structured and there is no way of pursuing an ‘intelligent’ approach to the information retrieval.

Nowadays many people surf the Web, but the task of searching information is left up mainly to the users. Browsing tools available to assist people in locating information resources on the Internet include WAIS (Kahle 91), Gopher (McCahill 92) and, more recently, Web browsers, which in the last years have provided easy access to all the information available on the Web, using manual navigation through hypertext links.

Among the many search engines, Altavista, Yahoo, Lycos and Google - which indexes more than one billion Web pages - are the most remarkable.

There is, however, a growing need to complement current browsing and keyword-based techniques with concept-based approaches. One solution uses some kinds of metadata to describe the content of the underlying data repositories. In particular, some authors advocate using ontologies as metadata (Mena and Illarramendi 2001).

Although this is the most welcome solution in the community, other ways to deal with the lack of structure in the World Wide Web cannot and should not be disregarded. The creation of a semantic web is certainly a top of the list priority, but in the

meantime the community has also tried to develop semantic search engines to surf the WWW as it is at present.

Both the ideas of Semantic Web and Semantic Search Engines are based on Word Sense Disambiguation. The words of all natural languages have a certain degree of lexical ambiguity, which can be either syntactic or semantic. Such ambiguity is a major disadvantage in the keyword based Information Retrieval methods.

The problem of syntactic ambiguity in Natural Language Processing has been solved by part-of-speech taggers, which are now able to associate to a word its correct syntactic category. A major example of syntactic tagger is Brill's algorithm (Brill '95).

On the contrary there is still a long way to go before solving the problem of Semantic Ambiguity. The problem is that words can have several meanings, sometimes even very different ones, so the meaning of a word can be pinpointed only through a close examination of the context in which the word itself occurs. This is a very easy task for human beings, but not for computers.

There are two major approaches to WSD— the *knowledge based* approach and the *corpus based* one.

The first type of approach utilizes knowledge bases or explicit lexicons such as WordNet or Roget's International Thesaurus.

The corpus based approach tries to disambiguate words by means of information gathered by training on a corpus.

Following the inspirational works by Moldovan and Mihalcea, the authors have devised a knowledge based disambiguation algorithm, which utilizes WordNet and AltaVista Search Engine. The use of WordNet for Word Sense Disambiguation will be presented in this paper. In the first part an overview of related works in the field of Word Sense Disambiguation is introduced. This part is followed by a presentation of an algorithm for disambiguation (part 2). The rationale of the algorithm is then introduced in part 3, which is followed by a description of an application example (4). Part 5 reports the results of this study and an outlook of future applications is finally introduced in part 6.

1 Overview

Word Sense Disambiguation (WSD) is considered very important by researchers in the computer science area. The fields in which it has shown a greater relevance are Information Retrieval, Cross-Language Information Retrieval and Automatic Translation.

Despite the efforts of the whole community, WSD is still a problem and much work remains to be done. As a matter of fact, algorithms and the applications based on them bring forth disambiguation results that are too low to be effective, namely useful, in any of the above mentioned fields. In other words, either the algorithms produce a percentage of correctly disambiguated terms which is too low, or they work only when dealing with particular grammatical categories.

A statistical approach to the problem is usually considered the best one. Heuristic approaches have also been tried, for example by Anil Chakravarthy at MIT in 1995,

but they have resulted ineffective, that is to say useless within the field of IR, as Sanderson has demonstrated (1994).

According to Sanderson and the rest of the community, in order to be effective for Information Retrieval, an algorithm should reach a very high level of accuracy (90%). On the other hand during the SENSEVAL competition, held in April 2000, the percentage of accuracy provided by the algorithms presented oscillated between 70% and 80%.

However, if we look at more recent SENSEVAL Competitions (Edmonds, Cotton 2002) we see that we are still far from reaching the effectiveness of a human NLP system. Such effectiveness is 96.8 % according to Gale, Church and Yarowski (1992). Yarosky (1992), Aguirre (1996) and Mihalcea (1999) among others have devised algorithms for WSD. Their algorithms can only sometimes disambiguate to a high degree of accuracy, although they may even reach peaks of 92%. Nonetheless they are quite limited in that their focus is only on Verb-Object or Noun-Verb word pairs. Furthermore, in some cases the algorithms have been tested only on few samples (12 polysemous words in the work by Yarosky 1992).

On the other hand, algorithms with a broader focus, such as that devised by Katz et alii (1998), provide result percentages which are too low (60%).

If we agree with Sanderson and with the prevailing ideas at present, it is quite evident that these results cannot be considered useful for Information Retrieval.

Basing their work on a statistical approach, the authors have thus devised a new algorithm which avails itself of the electronic database WordNet 1.6 and of the AltaVista Search Engine. The reason for choosing the 1.6 release of WordNet and the AltaVista search engine is that most previous works were based on them too, thus the result achieved could be better compared with the others'.

The main idea of this paper has been drawn from the many works by Mihalcea and Moldovan (1999 on) where WordNet glosses were used as a context for WSD. The problem in this kind of approach is that the glosses are too few and too short. As a consequence Mihalcea and Moldovan's algorithm can only deal with Noun-Verb word pairs, because WordNet glosses do not provide sufficient context for other categories such as adjectives and so on. The authors intended to create an algorithm that might deal with a greater number of grammar categories, but such algorithm is the product of an early stage of their research activity, so they haven't dealt yet with WordNet collocations and figurative language.

Anyway, in order to widen the number of categories to which the algorithm could be applied, the authors have decided to use the Internet as a repository of glosses. As a matter of fact the Internet provides a wider variety of examples and allows to broaden the use of the algorithm to all different kinds of word pairs.

In this work, then, WordNet has been used only as a Thesaurus so that by synonymy the search could be broadened not only to specific words, but to specific concepts also. The use of the Internet and WordNet, though, has imposed some limitations as well. Indeed the Internet can produce a lot of casual co-occurrences, and the number of synonyms in each WordNet synset varies remarkably. In order to reduce the influence of the two above mentioned phenomena on the final results, the authors have devised a specific procedure, which will be illustrated later on.

2 The Algorithm and How It Works

Our algorithm has been devised in order to disambiguate queries expressed in natural language. It is based on the conviction that frequency and co-occurrence of terms are the elements that allow a correct disambiguation in a human NLP system. Such conviction cannot be demonstrated, of course, but it is rather intuitive and has already been used in this same application field. For example the Lesk algorithm (1986) is based on the idea that words which co-occur in a sentence are used to refer to the same topic and, on the other hand, their particular senses should be defined in a dictionary using some of the same words.

Basing their work on the above idea, the authors have devised an algorithm that consists in ten steps. In order to facilitate the explanation of these steps, the authors will take into consideration cases in which only one of the terms is ambiguous.

The ten steps are as follows:

1. A phrase showing at least one ambiguous term is chosen.
2. The terms are looked up in WordNet to find the synsets that express their different senses.
3. The terms in each synset of the ambiguous word are coupled with the terms in the synset of the unambiguous word.
4. The couples of terms thus obtained are looked up in the Internet. To this purpose the authors have decided to use the Boolean operator NEAR and the Altavista advanced research mode.
The Boolean operator NEAR has not been used when dealing with an “adjective-noun” kind of couple. In this case the inverted commas have been preferred, instead.
5. All the terms in both synsets are looked up in the Internet one by one.
6. The number of hits obtained in step 4 is then divided by the sum of the hits of each single term (step 5) that forms the pairs analyzed in step 4.
7. The figures obtained are summed in order to obtain a score to be attributed to the sense of the ambiguous term. From now on the authors will refer to this score as Sense Score (SS).
8. The SS obtained is then divided by the terms of a monotone increasing function of the number of synonyms in the synset, thus obtaining a ‘Weighed Score’.
9. The steps from 3 to 8 have are repeated for all the synsets (i.e. the meanings) of the ambiguous term.
10. The sense marked by the highest Weighed Score is acknowledged as the correct one.

3 The Rationale of the Algorithm

Step 1 simply deals with the choice of two words one of which is ambiguous. It implies no difficulties because in reality the user of the algorithm uses the algorithm itself to disambiguate an ambiguous expression he has come across. The algorithm can also be used to disambiguate an ambiguous expression made of N terms, as the

tests carried out have shown. In any case the testing phase has dealt only with expressions made of up to 4 terms. Even when dealing with more than one ambiguous term at a time the algorithm has produced accurate results. The tests clearly show that the degree of accuracy is not negatively affected by the number of ambiguous words. On the contrary, the greater the number of words, the more accurate the results are.

In step 2 WordNet was used because it is the wider lexical database available on line for free. Besides this, it was the most common resource for previous research. Its use makes a comparison with algorithms devised by other authors possible and it also enables other users to test the algorithm illustrated in this paper.

Step 3 needs no explanation because it only consists in the pairing of the terms.

In step 4 the authors chose the Boolean operator NEAR, neglecting other operators. Two opposite possibilities were considered: two terms may occur close to each other in a document, in fact they may even occur one after another. Sometimes, though, the two terms may be distant from each other in a document. In this case it is not certain whether their location is sufficient to create a semantic relation by which the two terms may disambiguate each other.

The operator NEAR was thus acknowledged as a middle way in order to achieve a good degree of approximation.

In the case of adjective-noun pairs the operator NEAR was disregarded, in that adjectives in the English language are usually followed by nouns. In this case a query in inverted commas was favored. This procedure does not consider the cases in which the adjective is parted from the name by a verb (i.e. to be) or by another adjective. Nonetheless a large amount of cases can thus be captured by the above procedure, whereas the use of NEAR would make the user retrieve a lot of documents where the adjective is referred to some other noun, thus being useless to the purpose of disambiguation.

In step 6 the number of hits obtained in step 4 was divided by the sum of hits of each single term (step 5) forming the pairs, because it was necessary to differentiate the results according to whether the terms co-occurred more or less frequently. In this way the casual co-occurrence of such pairs within the same document has shown a lesser influence on the final result.

Because the pairs were formed with the terms contained in one synset, the sum (step 7) of the results obtained in step 6 represents the score (SS) of that same synset or, in other words, of one of the senses of the ambiguous word.

However, the above score is still affected by the number of terms included in the synset. It is indeed quite evident that a larger number of hits can easily give rise to a higher sense score. The fact that some synsets include many synonyms and some include only a few could not be disregarded. By dividing the SS by a monotone increasing function of the number of synonyms, more balanced results have been achieved (step 8). However, the balance thus achieved depends for the most part on the chosen monotone increasing function. Indeed the wrong function could have brought about too low results when dealing with terms that have a great number of synonyms.

Our function had to be increasing when the number of synonyms in the synset increased. If we consider the formula

$$WS = SS/F$$

we see that the function is the denominator of the fraction. Thus the Weighed Score decreases when the number of terms in the synset increases.

Besides, our function had to be $F > 1$ otherwise the Weighed Score would have increased instead of diminishing.

Among the many different monotone increasing functions, logarithmic functions have a peculiar property: their derivative tends to zero if the argument of the function increases.

This type of function sets a more remarkable difference for the intervals of numbers close to zero on the x-axis than for intervals which are distant from zero on the same axis

$$[f(n+1)-f(n)]$$

For example, the difference between 101 and 100 is much lesser than the difference between 11 and 10:

$$(f_{101} - f_{100} < f_{11} - f_{10}).$$

In other words

$$f(x+\varepsilon)-f(x) > f(kx+\varepsilon)-f(kx),$$

where k is an integer > 1 and ε is > 0 .

Because the number of synonyms in WordNet synsets is usually a small one, the authors deemed more appropriate emphasizing the differences between small numbers. So this type of function has seemed especially suitable.

Thus a logarithmic function to the base 10 has been chosen. The argument of our function is $N+9$ where N is the number of synonyms and 9 is added so that the value of the function is always ≥ 1 .

4 An Application Example

Let's now examine an example step by step so that the disambiguation process may be cleared up.

Step 1)

Apparent Solution is the chosen expression. Furthermore, in order to be brief and facilitate the reading of this paper, the authors have also chosen to disambiguate only the term *Apparent*, while the term *Solution* has been considered unambiguous, that is having only one synset. Besides its only synset comprises only the term *Solution*.

Step 2)

The synsets of the term *Apparent* as they have been retrieved in WordNet are as follows:

Apparent

1. **apparent, evident, manifest, patent, plain** -- (clearly apparent or obvious to the mind or senses; "the effects of the drought are apparent to anyone who sees the parched fields"; "evident hostility"; "manifest disapproval"; "patent advantages"; "made his meaning plain"; "it is plain that he is no reactionary"; "in plain view")
2. **apparent, ostensible, seeming** -- (appearing as such but not necessarily so; "for all his apparent wealth he had no money to pay the rent"; "the committee investigated some apparent discrepancies"; "the ostensible truth of their theories"; "his seeming honesty")
3. **apparent** -- (readily apparent to the eye; "angry for no apparent reason"; "had no visible means of support")

Step 3–4)

In this section the authors have coupled the unambiguous term with each term in each of the synsets. The synsets have been considered separately, that is one by one. Because we were dealing with a noun-adjective pair, we have looked such pair up in the Internet using inverted commas instead of the Boolean operator NEAR. The number that appears right of each expression is the number of *hits* of each single search.

First sense of Apparent

apparent solution	917
evident solution	107
manifest solution	19
patent solution	100
plain solution	109

Second sense of Apparent

apparent solution	917

ostensible solution	12
seeming solution	45

Third sense of Apparent

Apparent solution	917

Step 5)

The authors then searched each single term in the Internet. The list below shows the number of hits obtained for each single term research.

Single term research

Apparent	13293
Evident	2811438
Manifest	914204
Patent	1919463
Plain	5917993
Ostensible	52250
Seeming	234759
Solution	21780725

Step 6–7)

According to what described in step 6 and 7 the number of *hits* of each single pair was divided by the sum of the *hits* of each single term in the pair itself. Secondly the results thus obtained for each synset have been summed.

First synset

$$(917/(13293 + 21780725)) + (107/(2811438 + 21780725)) + (19/(914204 + 21780725)) + (100/(1919463 + 21780725)) + (109/(5917993 + 21780725)) = 5,5418e-5$$

Second synset

$$(917/(13293 + 21780725)) + (12/(52250 + 21780725)) + (45/(234759 + 21780725)) = 4,4669e-5$$

Third synset

$$917/(13293 + 21780725) = 4,2076e-5$$

Step 8–9)

We have then divided the score obtained by the terms of a monotone increasing function of the number of synonyms in the synsets, thus obtaining a ‘weighed score’.

First synset (weighed score)

$$5,5418e-5/1,15 = 4,8189e-5$$

Second synset (weighed score)

$$4,4669e-5/1,08 = 4,1360e-5$$

Third synset (weighed score)

$$4,2076e-5/1 = 4,2076e-5$$

Step 10)

The sense with the highest score is the correct one, which, in our case, is that expressed by the first synset. In other words, according to the above algorithm, the ambiguous term *Apparent* within the expression *Apparent Solution* means *clearly apparent or obvious to the mind or senses*, that is *apparent, evident, manifest, patent, plain*, as a reader would expect by common sense.

5 Some Considerations and Results

Better results could undoubtedly be achieved if it were possible to take into account not only the number of documents in which a string of terms occurs, but also the number of occurrences of that very same string in each retrieved document.

As a matter of fact the strings (or some of them) formed using the synonyms found in the correct synset may probably occur more than once in each retrieved document. Although this kind of procedure could increase the accuracy of the algorithm, the authors have not gone through it, because it is extremely difficult and lengthy if performed manually.

Despite the fact that WordNet is still the best lexical database available on line for free, its granularity is sometimes excessive. This fact may cause a wrong disambiguation, in that it forces the algorithm to choose a meaning which is only slightly different from the correct one. Although rarely, this happens when two or more synsets include only one term and the term is the same. In these cases, though, the algorithm is still useful, because it certainly discards most of the wrong senses.

Among other considerations, it should be noticed that in case the algorithm is unsuccessful, there is very little difference between the SS which it recognizes as correct and the really correct SS. This means that the algorithm doesn't disambiguate wrongly, instead it disambiguates only partially.

Although no software has been implemented for an automatic use of this algorithm, such possibility (both legally and technically) is being evaluated at present by the authors. At the moment a most likely opportunity would seem that of creating a web interface because the algorithm must be able to use both WordNet and Altavista Search Engine at the same time.

The algorithm has been tested on over 50 examples of ambiguous phrases and expressions. 92 % of the results obtained have been successful, though this percentage has reached 95 % when an adjective has been coupled with a noun.

Among the pairs that have produced better results are the expressions: *Capital Letter*; *Next Week*; *Mother Tongue*; *Liberation Movement*; *Energy Generation*; *Remove Appendix*; *Incomprehensible Appendix*; *Microscopic Plant*; *Soap Opera*; *Linux Server*; *Obvious Solution*; *Drinkable Solution*.

6 Conclusions

We have already discussed the usefulness of WSD for Information Retrieval in the introduction of this work.

As for the creation of a 'semantic' Search Engine, the algorithm should be sided by other tools. As a matter of fact, it should be preceded by a system capable to turn a query phrased in natural language (NLQ) into a Boolean query (BQ) and followed by a query expansion system.

Furthermore, if it were possible to use a lexical database that includes terms belonging to more than one language, the algorithm described in this work could also be used in Cross Language Information Retrieval applications. Such algorithm would produce better results if WordNet synsets were widened as to include synonyms in other languages, in that the number of documents retrieved on the Internet would be much greater, so there would be more context provided for disambiguation purposes.

Acknowledgements. The authors wish to thank Dr Luciana Bordoni (ENEA-UDA/Advisor) for her precious advice and help in the draft of this work and for her patience in leading them into the field of computational linguistics studies. She has also provided a lot of material so that this study could be carried out.

The authors also wish to thank Dr Daniela Alderuccio for sharing with them her expertise on WordNet through long and inspiring conversations; thanks is to be also given to Dr. Sara Spaccini and Ada Nardin for their precious help in performing the above mentioned lengthy and boring tests.

References

1. Agirre, E., Rigau, G., "Word Sense Disambiguation Using Conceptual Density", in Proceedings of COLING : 16–22, 1996.
2. Berners-Lee, T., Cailliau, R. Luotonen, A. Nielsen, H.F., Secret, A., "The World Wide Web". *Communications of the ACM*, 37(8), August, 1994 : 76–82.
3. E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–566, December 1995.
4. Chakravarthy Anil S., "Sense Disambiguation Using Semantic Relations and Adjacency Information" in Proceedings of the 33th ACL : 293–295, Cambridge (MA), 1995.
5. Edmonds, P., Cotton, S. "SENSEVAL-2 Overview", in *Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems*. Toulouse, France.
6. Gale, Church, Yarowski. "Estimating Upper and Lower Bounds on the Performance of Word Sense Disambiguation Programs" in *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*: 249–256, Columbus, Ohio, 1992.
7. Kahle, B., Medlar, A., "An Information system for corporate users: Wide-area information servers", *Connexions: The Interoperability Report*, 5(11), November, 1991.
8. Katz, B., Uzuner, O., Yuret, D., "Word Sense Disambiguation For Information Retrieval", 1998, <http://www.ai.mit.edu>.
9. M. Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from a ice cream cone. In Proceedings of SIGDOC '86, 1986.
10. Litkowski, K., "Sense Information for Disambiguation: Confluence of Supervised and Unsupervised Methods", *ACL-02*, 2002, www.clres.com/online-papers/siglex02-feats.pdf
11. Magnini, B., Strapparava, C., Pezzulo, G., Gliozzo, "A. Using Domain information for Word Sense Disambiguation" in *Association for Computational Linguistics SIGLEX Workshop* Toulouse, France, 2001 : 111–4).
12. McCahill, M., "The Internet Gopher protocol : "A distributed server information system.", *Connexions: The Interoperability Report*, 6(7), July, 1992.
13. Mellish, C.S., *Computer Interpretation of Natural Language Description*, Ellis Horwood, Chichester (UK), 1985.
14. Mena, E., Illarramendi, A., *Ontology-Based Query Processing for Global Information Systems*, Kluwer Academic Publishers, 2001.
15. Mihalcea, R., and Moldovan, D., "A method for Word sense Disambiguation of unrestricted text", in Proceedings of the 37th annual meeting of the Association for Computational Linguistics (ACL-99), Maryland, 1999.

16. Mihalcea, R., Mihalcea, S., "Word Semantics for Information Retrieval: Moving One Step Closer to the Semantic Web", 2001, www.utdallas.edu/~rada/papers/ictai.2001.ps.
17. Mihalcea, R., Moldovan, D., "An Iterative Approach to Word Sense Disambiguation", 2000, <http://citeseer.nj.nec.com/mihalcea00iterative.html>
18. Mihalcea, R., Moldovan, D., "Pattern Learning and Active Feature Selection for Word Sense Disambiguation", in *Association for Computational Linguistics SIGLEX Workshop* : 127-30). Toulouse, France, 2001.
19. Mihalcea, R., Moldovan, D., "Word Sense Disambiguation with Pattern Learning and Active Feature Selection", *Journal of Natural Language Engineering* (s.d.).
20. Miller, G., "Wordnet: An On-Line Lexical Database", *International Journal of Lexicography*, 3(4) : 235–312, 1990.
21. Moldovan, D., Mihalcea, R., "A WordNet-Based Interface to Internet Search Engines", 1998, www.seas.smu.edu/~rada/papers/flairs98.ps.gz
22. Moldovan, D., Mihalcea, R., "Using WordNet and Lexical Operators to Improve Internet Searches", *IEEE Internet Computing*, vol.4 no.1, : 34–43, 2000.
23. Norvig, P., "Inference in Text Understanding", *Proceedings of the Sixth Conference on Artificial Intelligence*, Seattle, (Wa), 1987 : 561–565.
24. Resnik, P., "Selectional Constraints: An information-theoretic model and its computational realization", *Cognition*, 61, : 127–159.
25. Resnik, P., "Semantic Similarity in a Taxonomy: An Information Based Measure and its Application to Problems of Ambiguity in Natural Language", *Journal of Artificial Intelligence Research*, 1998.
26. Sanderson, M., "Word Sense Disambiguation and Information Retrieval", in *Proceedings of SIGIR '94*, 1994
27. Stetina, J., Kurohashi, S., Nagao, M., "General Word Sense Disambiguation Method Based on a Full Sentential Context", in *proceedings of COLING-ACL Workshop*, Montreal, Canada, 1998.
28. Yarowsky, D., "Unsupervised Word Sense Disambiguation Rivaling Supervised Methods", in *Proceedings of the 32nd Annual Meeting of the ACL*, 1995.
29. Yarowsky, D., "Word Sense Disambiguation Using Statistical Models of Roget's Categories Trained on Large Corpora", in *Proceedings of COLING '92*, Nantes, France, 1992.
30. Yuret D., *Discovery of Linguistic Relations Using Lexical Attraction*, Ph.D. Dissertation, MIT, 1998.
31. Zechner, K., "Fast Generation of Abstracts from General Relevant Sentences". *Proceedings of the 16th International Conference on Computational Linguistic, (COLING '96)*, 1996 : 986–989.

JTRES 2003 PC Co-chairs' Message

Workshop focus. Over 90 percent of all microprocessors are now used for real-time and embedded applications, and the behavior of many of these applications is constrained by the physical world. Higher-level programming languages and middleware are needed to robustly and productively design, implement, compose, integrate, validate, and enforce real-time constraints along with conventional functional requirements and reusable components.

Although it can serve as a basis for well-crafted software, Java is unsuitable for developing real-time embedded systems, mainly due to under-specification of thread scheduling and the uncertainty introduced by its garbage collection. Recently, to address these problems, a number of extensions to Java have been proposed, the two most representative being the Experts Group Real-Time Specification (RTSJ) for Java and the J-Consortium Real-Time Core Extension (RTCore). The intent of these specifications is the development of real-time applications by providing several additions, most notably those necessary to achieve predictable execution and to determine if deadlines can be met.

There is an increasingly growing interest in Real-Time Java in both the research community and in industry, because of its challenges and its potential impact on the development of embedded and real-time applications. The goal of this workshop is to foster interaction among researchers working on real-time and embedded Java, to identify the remaining challenges in Java for real-time, to assure the success of Real-Time Java as a technology, and to report the results and experiences of researchers.

Workshop contribution. Even in its first offering, JTRES was very successful in attracting researchers' attention. This is a tribute to the interest and potential for Java technologies in real-time and embedded systems. The papers included in this proceedings provide advances in several areas, and cover topics such as extensions to the real-time Java memory management and scheduling, software engineering techniques for real-time and embedded Java programming, performance analysis, and new processors and virtual machines for embedded Java solutions. Some papers provide good insight into future trends, while others make a clear point on the state of the art of Java technologies in real-time embedded systems.

August 2003 Angelo Corsaro, Washington University, St. Louis, MO, USA
Ron K. Cytron, Washington University, St. Louis, MO, USA
Corrado Santoro, University of Catania, Catania, Italy
(JTRES 2003 Program Committee Co-chairs)

Towards a High Integrity Real-Time Java Virtual Machine

Hao Cai and Andy J. Wellings

Department of Computer Science, University of York, UK
{haocai, andy}@cs.york.ac.uk

Abstract. This paper defines a run-time architecture for a Java Virtual Machine (JVM) that supports the Ravenscar-Java profile (RJVM). This architecture introduces an early class loading and verifying model that can facilitate the predictable efficient execution of Java applications, detect program errors at the initialization phase and prevent errors occurring during the mission phase. A pre-emptive fixed priority scheduler is provided which supports the immediate ceiling priority protocol to facilitate efficient resource usage. An economical predictable memory management model based on heap, immortal and linear time scoped memory (LTM) is presented. The integration of our proposed runtime architecture and the KVM is discussed.

1 Introduction

The combination of object-oriented programming features, the lack of any mechanisms for memory reclaiming (thereby necessitating the virtual machine to undertake garbage collection) and the poor support for real-time multi-threading are all seen as particular drawbacks to the use of Java in high integrity real-time systems (HIRTS). The Real-Time Specification for Java [1] (RTSJ) has introduced many new features that help in the real-time domain. However, the expressive power of these features means that very complex programming models can be created, necessitating complexity in the supporting real-time virtual machine. Consequently, Java, with the real-time extensions as they stand, seems too complex for confident use in high-integrity systems. Ravenscar-Java [3][4][5] has been proposed in order to counter these effects.

1.1 Ravenscar-Java

A Ravenscar-Java program consists of a fixed number of schedulable objects (real-time threads and asynchronous event handlers) that are created immediately after the program begins its execution. A program is considered to have

- an initialization phase – where all classes are loaded and all the schedulable objects (and all other permanent objects) are created; typically there are no hard time constraints associated with this phase of execution,
- an execution phase – where all schedulable objects execute under time constraints.

Schedulable objects in Ravenscar-Java do not terminate and, hence, the program does not terminate. All schedulable objects have unique fixed priorities and they are executed using pre-emptive priority-based scheduling. Sporadic entities are released by a single event that can be either software generated or hardware generated (by an interrupt).

Communication between schedulable objects is via shared data. Where data cannot be accessed atomically, it is encapsulated by synchronized methods that enforce mutual exclusion. Ceiling priority inheritance (called priority ceiling emulation by the RTSJ) is used to bound the time that a high-priority schedulable can be blocked by a low-priority schedulable object accessing shared data. Ravenscar-Java requires predictable memory management, hence, only LTM can be used for dynamic object creation.

1.2 J2ME

To address the demands of embedded systems and consumer electronics, the Java 2 Platform Micro Edition (J2ME) [7][8] has been introduced by Sun. This defines the three layered architecture illustrated in Fig. 1 [4][8]:

- a virtual machine layer which usually is implemented on top of a host operating system,
- a configuration layer which defines the set of Java language features, a minimum set of virtual machine features and the available class libraries that can be supported by a particular implementation platform (for example a mobile phone),
- a profile layer which defines a minimum set of Application Programmers Interfaces (API) targeted at a particular application domain.

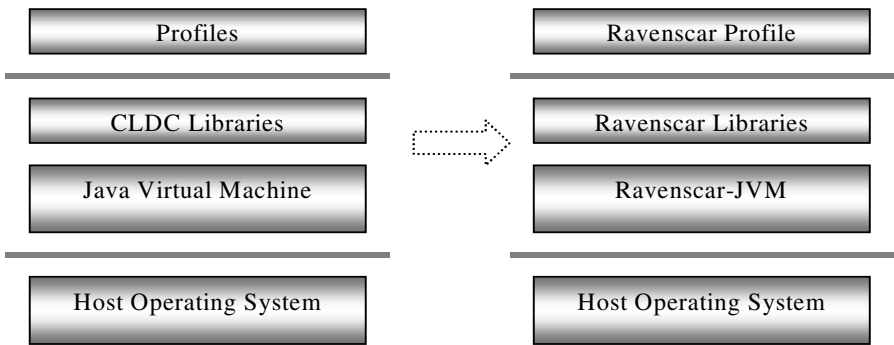


Fig. 1. The architecture of J2ME and Ravenscar-Java

A configuration layer, called Connected, Limited Device configuration (CLDC) [8] has been defined for small, resource-constrained mobile devices (mobile phones, pagers, personal organizer etc.) typically with a memory capacity of up to 512 KB. The K (kilo bytes) virtual machine (KVM) is a virtual machine specifically designed to support the CLDC. The restrictions imposed on the Java language and the virtual machine include: no support for floating point operations, no native interface, no user-

defined class loaders, no thread groups and daemon threads, no object finalization, etc.

Ravenscar-Java can be considered as a profile layer for the high integrity real-time application domain. It is supported by a Ravenscar-JVM (RJVM) which is based on KVM. The RJVM can not only preserve the portability of KVM but also targets the temporal non-deterministic execution problems of KVM. The Ravenscar-Java profile defines a set of APIs for HIRT Java applications and also defines a reliable and analyzable computational model. This results in the following restrictions on the RJVM:

- No dynamic class loading during the application execution phase,
- Pre-emptive priority-based scheduling with immediate ceiling priority protocol (ICPP),
- No garbage collection during the execution phase.

This paper discusses how the KVM can be modified in order to support Ravenscar-Java. Section 2 analyses the run-time architecture of KVM and identifies which features are unsuitable. A run-time architecture for the RJVM is then proposed, in Section 3, which targets these problem areas. Sections 4 and 5 describe a prototype implementation of the architecture and an experimental evaluation of its performance. Finally, our conclusions and future work are given in Section 6.

2 The Run-Time Architecture of K Virtual Machine (KVM)

In this section, the run-time architecture of KVM is presented. The three main unpredictable features of KVM: the dynamic class loading and verifying model, the threading model and the memory management model are discussed in detail.

2.1 The Class Loading Model

The KVM supports a dynamic class loading, linking and verifying execution model, illustrated in Fig. 2. Each instance of the KVM has a class loader that is responsible for loading the classes of an application. It then passes the loaded data to a linker, which parses the constant pools, fields, methods and other attributes contained in the loaded classes into the run-time class structures. The execution engine of KVM, which is an interpreter, starts as long as the essential classes (this includes some essential system classes and the main class) are loaded. When executing the application, the interpreter may access some instructions that refer to the constant pool that contain symbolic references. The resolver asks the class loader to dynamically load the referenced types and replace the symbolic references in the constant pool to direct references. When the interpreter first uses a class, it needs to be verified. Substantial effort is needed to verify every Java byte code of all methods in the class to ensure their type safety at run time.

The execution of instructions resulting in dynamic class loading and class verifying, are some of the main sources of unpredictability and inefficiency in the execution of Java applications. To increase predictability and efficiency, these dynamic loading and verification features should be removed.

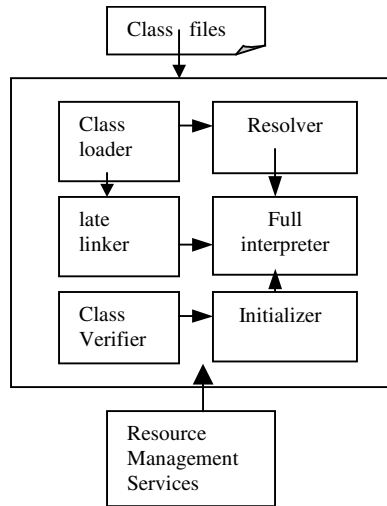


Fig. 2. A dynamic class loading and verification run-time architecture in Java

2.2 The Threading Model

KVM has a simple pre-emptive, variable quantum, round-robin scheduling model, in which all the active threads are stored in a circular linked list. It supports 10 priority levels. When a new thread arrives, it is linked to the end of the list and is scheduled to run. Each thread in the list is given a priority, which indicates to the interpreter the maximum number of bytecodes the thread may execute during its quantum. Each thread also has four runtime variables (a stack pointer, a frame pointer, a local pointer and an instruction pointer) and a stack which store its execution context. When a thread comes into existence, its runtime context is loaded into the four virtual registers of the interpreter: and also its quantum (timeslice) is set to $1000 \times \text{its priority}$.

When a running thread is blocked for a period of time, it is put into a timer queue. The threads in the timer queue are sorted by their wakeup times. When a thread switch occurs, first, the timer queue is checked from its first thread. The threads in the timer queue with their wakeup times due are moved from the timer. If a thread in the timer queue was blocked inside a monitor, the thread will be put into the wait queue for the monitor lock. If the thread in the timer queue was blocked by sleeping for a period of time, it will be put into the runnable queue.

KVM has an approach that attaches real monitor objects to object instances only when they are really needed. The header of an object instance stores information about the monitor that is associated with that object. There are four possible types of access to a synchronized object. First, no synchronization is required for the object. Second, the object is locked and accessed by only one thread once. Third, the object is locked and accessed by only one thread multiple times. Fourth, the object is locked by one thread and is requested by other threads. Only the latter requires a real lock to be used. Consequently, when multiple threads try to execute synchronized method calls

or MONITORENTER bytecodes, a monitor object illustrated in Fig. 3 is created to attach to the associated object.

Each monitor contains a pointer that refers to the current holder of the monitor and also holds a wait queue and a condvars queue in which those threads are unconditionally or conditionally waiting for the monitor to be released. When a thread tries to access an object associated with a monitor, a check will be performed. If the thread holds the monitor, it continues to execute. If the monitor is held by another thread, this thread is queued on the wait queue. A first-come-first-serve scheduling model is used for controlling the wait queues. A thread may request a timeout whilst trying to acquire a monitor lock, this thread is first put into the timer queue and the condvars queue of the monitor. After the period, the thread is moved to the wait queue.

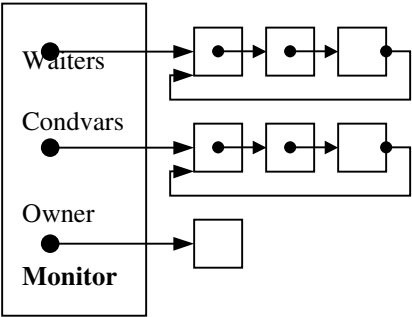


Fig. 3. The structure of a monitor [10]

2.3 Analysis of the Threading Model in KVM

All runnable threads are put in the runnable thread queue and are scheduled by the policy “first come first served”. The priority just simply enables the high priority threads to get a larger quantum. Priority inversion may frequently occur.

Table 1. Example Thread set for KVM.

Thread	Priority	Execution sequence	Release Time (instructions)
a	1	EQQE	0
b	2	EEEE	2000
c	3	EQQEE	2000

To illustrate priority inversion in KVM, consider the executions of three threads: *a*, *b*, and *c*. Assume they arrive in the order of *a*, *b*, *c* and thread *a* and thread *c* share the resource (synchronized object), denoted by the symbol *Q*. The priority of thread *c* is the highest and that of thread *a* is the lowest. Table 1 gives the details of the three threads and their execution sequences; in this table ‘E’ represents the execution of 1000 bytecode instructions and ‘Q’ represents the execution of 1000 instructions whilst holding *Q*’s lock.

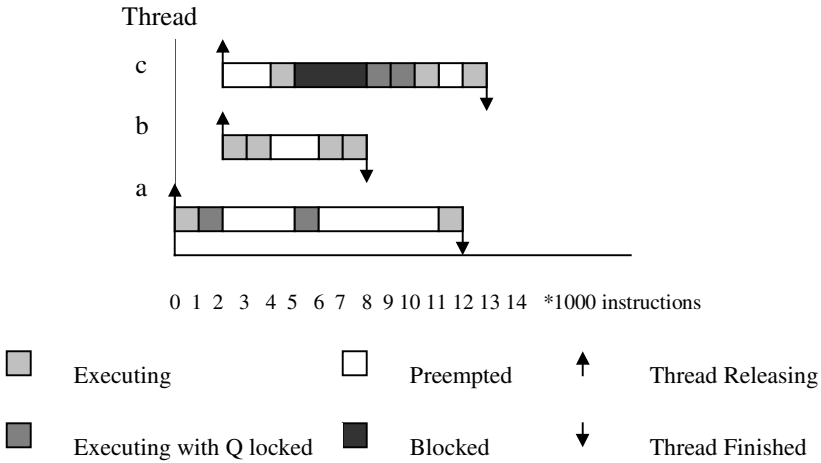


Fig. 4. Example of priority inversion with a shared resource in KVM

Figure 4 illustrates the execution sequence for the three threads. The quantum of threads are decided by their priority, so thread *b* has the double quantum of thread *a*, thread *c* has triple quantum of thread *a*. At each turn of running, thread *a* will execute 1000 instructions, thread *b* will execute 2000 instructions and thread *c* will execute 3000 instructions. Thread *a* is released first, executes and locks *Q*. Thread *b* and *c* are released at the same time. However, *b* arrives earlier than *c*, so *b* is scheduled to execute and finishes its 2000 instructions. Thread *c* is scheduled to run, it executes for 1000 instructions and then it is blocked when it tries to acquire the lock of *Q*, which is currently held by thread *a*. At the second iteration of the three threads, thread *a* executes 1000 instructions, releases *Q* and it is pre-empted by thread *b*. Thread *b* executes another 2000 instructions and finishes at time 8000. Thread *c* is scheduled to run, it acquires the lock of *Q* and executes its 3000 instructions. It is pre-empted by *a* after it finishes its quantum. At the third iteration of running, thread *a* executes another 1000 instructions and finishes at time 12000. Thread *c* then finishes its last 1000 instructions at time 13000. Priority inversion is very severe in that the lower priority threads *a* and *b* both finish before the highest priority thread *c*.

The threading model in KVM has the disadvantages of severe priority inversion and frequent context switches. It is not suitable threading model for high-integrity real time systems.

2.4 The Memory Management Model

The KVM uses a simple traditional garbage collection (GC) algorithm called Mark-and-Sweep collection [10][12]. At run time, the KVM is initialized with a chunk of memory called the heap. Stage 1 in Fig. 5 illustrates the status of an initialized heap. The newly created objects are allocated from the heap during program execution. Each object has a 32-bit header of which 24 bits is for the object size, 6 bits is for lock types and 1 bit is for the mark. Stage 2 in Fig. 5 illustrates the status that some objects have been allocated in the heap. When the heap is full, the GC starts to mark the

objects in the heap. It starts marking the global root objects that consist of all the loaded classes, all threads and all thread stacks and some temporary root objects. For each live object, the GC recursively marks all those objects that are reachable from it. After marking, the GC sweeps the heap and all the free memory spaces are linked into a list of small chunks which is illustrated in stage 3 of the heap in Fig. 5. The linked chunks are used for later memory allocation. If the largest chunk is not large enough for a memory request, a compaction occurs. This moves all the live objects together, upgrades pointer in the live objects and links all the free small chunk into a large free block, then the large block is used for later memory allocation.

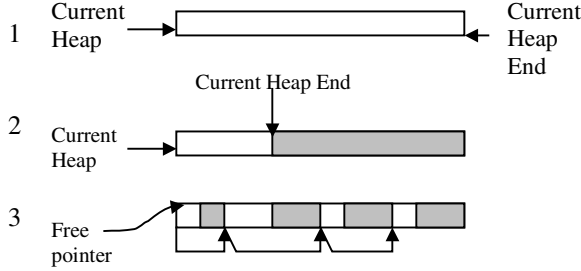


Fig. 5. The three stage of a heap in KVM

The mark and sweep GC in KVM traces live objects in the mark phase and sweeps and compacts linearly throughout the entire heap. Let M be the size of the heap, R be the amount of live objects and a, b be constants. The time complexity of the mark-sweep collector can be approximated by [12]:

$$t = aR + bM. \quad (1)$$

The amount of space recovered by a garbage collection is:

$$M = M - R. \quad (2)$$

Define the efficiency, e , as the amount of memory reclaimed in a unit time [12]

$$e = \frac{M - R}{aR + bM} = \frac{1 - r}{b + ar}, \quad (3)$$

where $r = R/M$ is the residency of the program. Let gc be the amount of time to perform garbage collection (GC), $overall$ be the execution time of the applications and c be the allocation rate (memory allocated in a unit time). Define the GC overhead, o , as:

$$o = gc / overall = \frac{c \cdot \frac{1}{e} \cdot overall}{overall} = \frac{c}{e} = c \cdot \frac{b + ar}{1 - r}. \quad (4)$$

The mark and sweep GC in KVM has the disadvantage that it is not efficient and is highly unpredictable. The larger the heap, the longer the GC pauses as shown in

equation (1). This is because the larger heap the more live objects it holds and the larger the space that needs to be swept and compacted. The behavior of GC is highly unpredictable and it depends on the status of virtual machine and dynamic features of Java programs, for instance, the residency of the program, r shown in the equation (3)(4).

3 Run-Time Architecture of a RJVM

We have discussed the problems of adapting KVM so that it supports Ravenscar-Java. Dynamic class loading and verifying provide enormous uncertainty for the application's execution. The thread model has frequent context switches and suffers from priority inversion which prevents efficient resource usage. The high overhead of GC and long GC latency make it unsuitable for HIRTS. In this section, we will present our solutions to these problems. A new class loading and verifying model, a real-time thread model and a predictable memory management model for a RJVM are proposed.

3.1 A Class Loading and Verifying Model of a RJVM

In the initialization phase of a Ravenscar-Java application, a static `NoHeapRealTimeThread` executes first and performs the initialization of the application before the mission phase is carried out. It includes initialization of all real-time threads, memory objects, event handlers, events and scheduling parameters. In the mission phase, the application is executed and multithreading is dispatched based on the imposed scheduling policy. It is difficult to have a reasonable upper bound execution time for the Java primitive operations that may result in dynamic class loading and verifying during the application's mission phase. The Ravenscar-Java requires that all the classes needed in the application be loaded during the initialization phase.

A new run-time architecture is proposed to support this illustrated in Fig. 6. All the classes needed by the application are loaded, linked into the RJVM and also are verified at the initialized stage of the application. The virtual machine starts to load the essential system classes and then loads the main class of the application. All the classes that can be reached from the main class will be loaded before the interpretation. During linking classes, not only the syntax of classes but also every Java byte code of every method in the classes is verified [9][15]. During the verification of each Java byte code, the instructions that need dynamic resolutions of the constant pool entries are replaced with their correspondent fast instructions and these constant pool entries are resolved. For example, when verifying a new instruction, the constant pool entry requires that the new instruction will be resolved and the new instruction be replaced with `new_quick` instruction. During the execution stage of the application, no dynamic loading, linking and verification will be performed. All the instructions that may invoke dynamically loading and linking classes are replaced with their correspondent quick instructions. No dynamic verification of classes will be needed during the interpretation of Java byte code.

The early loading by the verifier simplifies the interpretation of Java byte code. All the instructions associated with dynamic class loading such as `getfield`, `new`, `invokevirtual` etc. can be replaced with their correspondent quick instructions such as `getfield_quick`, `new_quick`, and `invokevirtual_quick` etc. 24 slow instructions can be removed from the instruction set of the interpreter. This model has the advantages that:

- The instruction replacement is performed at the initialization phase of the execution, so it is backward compatible with other Java applications for KVM and performance and predictability are greatly improved.
- The simplified instruction set can facilitate more accurate worst case execution time (WCET) analysis of Java bytecode execution [6].
- The Java primitive operation, which may result in dynamic class loading and verification, has been replaced with their correspondent quick instructions and can be given a reasonable upper bound WCET.
- Comprehensive type safety checks become possible because all the related classes are available for the verifier before the mission phase.

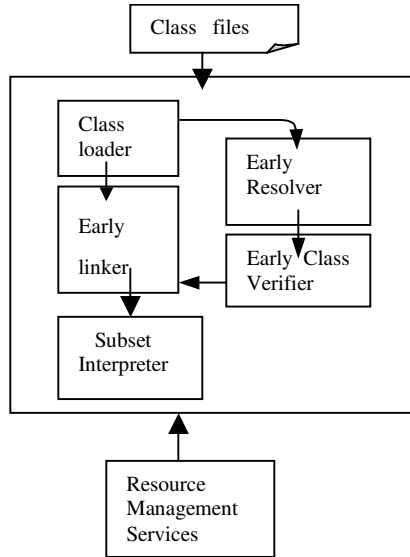


Fig. 6. A early class loading and verifying runtime architecture in RJVM

3.2 A Scheduling Model for a RJVM

To solve priority inversion problem in the KVM, the thread system and the lock system need to be modified. Inside our RJVM, instead of a single runnable queue, a list of runnable queue is created and each priority has a runnable queue [16]. The scheduling of threads starts from the highest priority queue to the lowest priority queue. The scheduling policy of the same priority queue is pre-emptive round-robin

scheduling. The newly created threads are put into the end of their correspondent runnable queue. The next thread to run when a context switch occurs is the head of the non-empty highest priority queue.

The monitor wait queue also needs to be modified from a “first come first serve” queue to a priority-based queue. The highest priority threads in the wait queue of a monitor will hold the monitor instead of the first waiting thread when the owner releases the monitor. The immediate ceiling priority protocol [13][14] is also provided to avoid the priority inversion when shared resources are involved between threads.

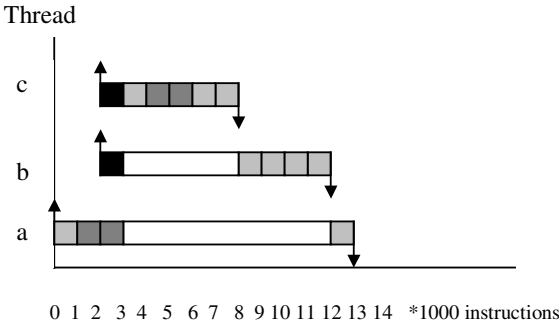


Fig. 7. Example of modified thread system in KVM

With the new scheduling, the execution sequence of the three threads in Fig. 4 will change to that illustrated in Fig. 7. The shared resource Q will be given a ceiling value of 4, which is higher than the maximum priority of processes that use it. When thread *a* is locking the resource Q, thread *a* inherits the ceiling value of resource Q, here thread *a* is scheduled to run with dynamic priority 4. This ensures that thread *a* is not pre-empted by the threads with priority 3 when thread *a* holds lock of Q, so *a* can finish and release Q as soon as possible. When thread *a* releases the lock of Q, it restores its previous dynamic priority and thread *c* acquires the lock of Q and is scheduled to execute. After *c* finishes, *b* is scheduled to run and finish, then *a* executes and finishes. No priority inversion occurs.

3.3 A Predictable Memory Management Model for a RJVM

A predictable memory management model based on heap, immortal and LTM [2][5] is shown in Fig. 8. This model addresses Ravenscar-Java concerns for efficiency and predictability and yet the programmer can benefit from the Java object-oriented programming style while avoiding the side effect of garbage collection. At the initialization phase of a virtual machine, a heap and a block of memory for linear time memories (LTM) are allocated. The heap is used to allocate objects that include the loaded classes, the String objects and some temporary objects etc. During this phase, the heap is subject to garbage collection. Just before the start of interpreting, the heap is compacted and is transferred to immortal memory, which is not subject to garbage collection. During the initialization of Java applications, all new created objects including LTM objects are created in immortal memory. However, the memory space

that LTM objects refer to are allocated from the memory for LTMs. During the mission phase, all the new created objects are allocated either in immortal memory or linear time memory which are not subject to garbage collection. Each thread is associated with a memory area, either a LTM or the immortal memory area at any instance. When a thread is scheduled for execution, its memory area will become the current memory area and all objects dynamically created are allocated in the area during the program execution.

This approach has the advantages of more efficient and predictable memory usage compared with the model of KVM. The garbage created at the initialization phase can be removed by using a traditional garbage collected heap. A substantial memory could be wasted without collecting garbage objects created during the loading of classes. By the early loading of classes and collecting and compacting the heap at the end of the initialization phase, this memory will be saved for use by the application. Experiments indicate that the memory space saved is around 1.3 times of the application size. The early loading of all classes can facilitate the efficient memory usage because a large amount of garbage objects can be collected after finishing the class loading. For devices with limited memory, the collection of the heap is essential during the initialization phase of the application.

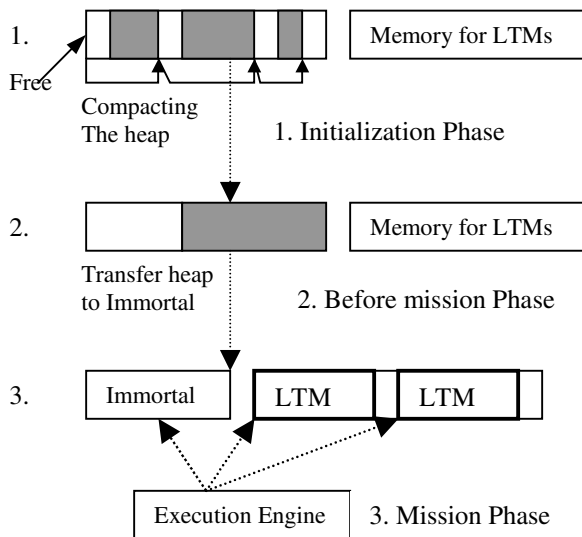


Fig. 8. A predictable memory management model

During the execution phase, a linear time memory can be associated with and periodically reused by a periodic thread. The reason for separating the memory for LTMs from the immortal memory is to ease the runtime assignment check. It will become much easier to tell whether an object created is in the immortal or in the LTM block. This can simplify the runtime to check that an object created in LTM is not assigned to objects created in immortal memory or heap.

4 Implementation Issues

In this section, the integration of our proposed runtime architecture and KVM is illustrated in detail.

4.1 Implementation of ICPP

The Ravenscar-Java profile requires ICPP to be implemented in the run-time system. KVM has a lightweight monitor system discussed in 2.2 that creates real monitor objects to object instances only when they are really needed. To implement the ICPP, one attribute (`int ceilingPriority`) is added to each shared resource lock, not to every object and two attributes (`int org_priority[MaxObjectLocking]`, `int CeilingCount`) are added to the thread class. Integer `ceilingPriority` holds the ceiling value of a shared resource which is greater than the maximum priority of the threads that can use it. An integer array `Org_Priority` stores a list of the ceiling value that the thread is inherited from the shared resources. `CeilingCount` is the number of the priority inheritance occurrences.

Each thread starts with its static default priority as its dynamic priority. When it tries to acquire an available shared resource, the ceiling priority of the resource is compared with the thread's dynamic priority. If it is higher than the thread's dynamic priority, priority inheritance occurs, the thread's current dynamic priority is stored into the `org_priority` array and the number of the `CeilingCount` is increased. The thread's current dynamic priority inherits the ceiling priority of the shared resource. The thread will execute with the dynamic priority until it releases the shared resource. When the thread releases the resource, its previous dynamic priority is restored and the `CeilingCount` is decreased. If a thread is trying to acquire a shared resource whose ceiling priority is lower than the thread's dynamic priority, a monitor holding error will occur. Ravenscar-Java does not allow a thread holding a higher dynamic priority to enter a lower ceiling priority monitor.

Two attributes (`int NoLock`, `int NoLockCount`) are also added to each runtime thread and one attribute (`int NoLock`) is also added to the resource monitor to support `NoLockPriorityCeilingEmulation` that prevents a thread suspending when it is holding `NoLock` shared resources. When a thread is holding a `NoLock` resource, the thread `noLock` flag is set to 1 and its `noLockCount` is increased. When the thread releases the `noLock` resource, the `noLockCount` of the thread is decreased. The `NoLock` flag of the thread will set to 0 when its `NoLockCount` becomes 0. A thread with the `NoLock` flag set to 1 is not allowed to suspend.

4.2 Implementation of the Memory Management Model of RJVM

Ravenscar-Java only defines one type of scoped memory area, Linear Time Scoped Memory (LTM) and it also requires that access to LTM areas must not be nested and LTM areas must not be shared between schedulable objects. All threads may use immortal memory or LTM during the run time. This implies that each thread can only

be associated with one memory area, either a LTM or the immortal memory at any instance. The run-time instance of threads is extended and the runtime instance of memory area is created. Their data structures are illustrated in Fig. 9.

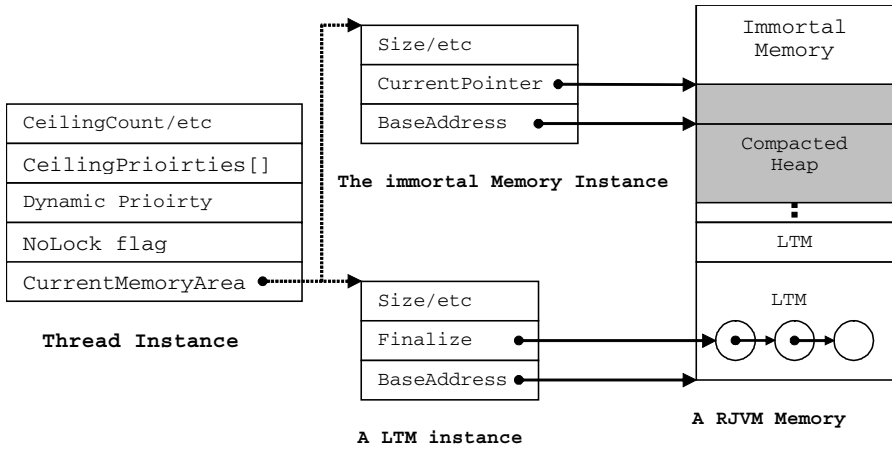


Fig. 9. Runtime structures of thread and memory Area

Each runtime thread is extended to contain a memory area pointer (MemoryArea*) which points to the current memory area of each thread. The thread uses this to hold new created objects. The current memory area could be the immortal memory or a LTM. Each RJVM has only one immortal memory while multiply LTM areas could be created in the memory for LTMs. Each run-time memory area contains the information such as its maximum size, the free memory size, the pointer to a block of memory allocated from the immortal memory, the current free pointer that contains the address of the free memory etc.

Each LTM also contains a list of finalizable objects. When an object of a class which contains a finalize method is created in a LTM, the object will be put in the finalizable list of the LTM. When the memory of a LTM is to be reclaimed, the objects in its finalizable list are finalized first, then the LTM is restored to its initial state such as the current pointer points to its base address, remaining size is restored to the initial size of the LTM, the finalizable list is null etc. The LTM then can be reused.

5 Experiments Evaluation

This section evaluates the changes made to the KVM to form the basis of a RJVM. To evaluate the policy of collecting memory after loading all classes just before executing the application, four small applications are used. The size of the applications, memory collected and the ratio of application size with memory saved are illustrated in table 2. By compacting the heap before the start of the main method, around 1.3 times the application size memory could be saved before the mission phase of the applications.

Table 2. Memory reclaimed after loading all classes before the mission phase

Size of applications	Garbage collected	Ratio
2169	2976	1.37
2341	3128	1.33
4162	6196	1.34
5276	6884	1.30

A benchmark with three threads denoted by the symbol a , b and c is used to evaluate the thread model in KVM and in our RJVM. The priorities of the thread threads, their order arrived, their time released, their computation block in $1 * 10^6$ instructions and their microsecond response time both in KVM and in our RJVM are detailed in Table 3. No garbage collection occurs during the execution.

Table 3. Example thread set and their response time in KVM and RJVM

Thread	Priority	Order arrived	Time released	Computation block	Response time in KVM	Response time in RJVM
a	1	1	0	890	177846	177806
b	2	2	0	1780	177876	148194
c	3	3	0	2670	177896	88908

The scheduler in KVM has a very high context switch overhead which results in quite inefficient execution of the Java applications. It is not a suitable model for multithreads applications. Our RJVM provides a predictable and efficient multithreads execution environment.

The garbage collection benchmark used to evaluate the memory management in KVM and our proposed memory model is the tree benchmark which is an adaptation made by Boehm from the Ellis and Kovac benchmark [17]. It is modified to be compatible with the KVM. The allocation rate of the modified tree benchmark is about 13.75 K bytes per microsecond. The number of garbage collection pass, the microseconds spent in garbage collection, the microseconds spent in execution and the percentage overhead introduced by GC in KVM for this benchmark is illustrated

in Table 4. From the equation: $o = c \cdot \frac{b + ar}{1 - r}$ in section 2.4, we can understand the

behavior of GC in KVM shown in Fig. 10. Horizontally, for a single application, the smaller heap, the larger residency of the program, r , and then the larger GC overhead. Vertically, for different applications, the larger allocation rates, c , the larger GC overhead.

The behavior of the tree benchmark in our implementation of RJVM is illustrated in table 5. Predictable and efficient execution is achieved. During the mission phase, no garbage collection will occur. By separating immortal memory and linear time memory (LTM), it is relatively easier to check that an object created in LTM is not referenced by objects created in immortal memory. The runtime check overhead (RCO) is the amount of the time to perform runtime check with the execution time.

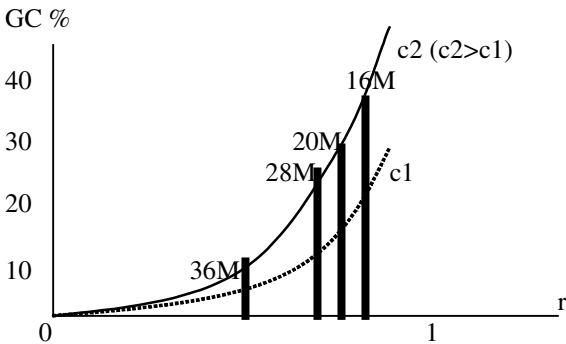


Fig. 10. The behaviour of GC in KVM

Table 4. The garbage collection overhead of the tree benchmark

Memory heap	GC pass	Collection time	Execution time	% overhead
16M	4	1020	3365	30.31
20M	3	1011	3355	30.13
28M	2	961	3325	28.90
36M	1	341	2694	12.66
64M	0	0	2424	0

Table 5. The execution time of the tree benchmark in the RJVM

Heap/Immortal	LTM	Execution Time without RC	Execution Time with RC	RCO %
16M	12 M	2504	2704	7.4

6 Conclusion

In this paper, a runtime architecture of RJVM has been presented. An early class loading and verifying model can detect the program errors at the initialization stage; it will reduce the failures of the system during the mission phase which might have catastrophic consequences. A more deterministic threading model reduces the context switch overhead of the KVM and facilitates efficient resource usage by adopting the immediate ceiling priority protocol. A memory management model based on heap, immortal and linear time memory provides efficient and predictable memory usage without sacrificing the automatic memory management features of Java. The experiments show great improvements on predictability and efficiency when executing Java applications in our RJVM compared with the KVM. A more reasonable upper bound WCET of Java Byte code can be obtained by removing the effect of garbage collection and dynamic class loading and verification.

References

- [1] G. Bollela, B.Brosgol, P.Dibble, S.Furr, J.Gosling, D. Hardin, and M.Turnbull, "The Real-time Specification for Java", Addison Wesley, 2000
- [2] G. Bollella, K. Reinholtz, "Scoped Memory", IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002
- [3] P. Puschner and A.J. Wellings, "A Profile for High-integrity Real-time Java Programs", IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001
- [4] J. Kwon, A.J. Wellings, and S. King, "Ravenscar-Java: A High Integrity Profile for Real-Time Java", Proceeding of the Joint ACM Java Grande – ISCOPE 2002 Conference, 2002
- [5] J. Kwon, A.J. Welling, and S. King, "Predictable Memory Utilization in the Ravenscar-Java Profile", IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003
- [6] G. Bernat, A. Burns, and A.J. Wellings, "Portable Worst-Case Execution Time Analysis Using Java Byte Code", Proceedings of the 12th EuroMicro Conference on Real-Time Systems, Stockholm, June 2000
- [7] Sun Microsystem®, Java™ 2 Platform, "Micro Edition (J2ME™) Technology for creating Mobile Devices", white paper, <http://java.sun.com/products/cldc/>, accessed March 2003
- [8] Sun Microsystems, "Connected, Limited Device Configuration, Specification Version 1.0a", <http://java.sun.com/products/cldc/>, accessed March 2003
- [9] T. Lindholm and F.Yellin, "The Java Virtual Machine Specification(2e)", Addison Wesley, 1999
- [10] F. Yellin,, "Inside the The K Virtual Machine (KVM)", Presentation slice, <http://java.sun.com/javaone/javaone2001/pdfs/1113.pdf> , accessed April, 2003
- [11] B. Venners, "Inside the Java Virtual Machine", McGraw-Hill, 1999.
- [12] R. Jones, R.Lins, "Garbage Collection-Algorithms for Automatic Dynamic Memory Management", John Wiley & Sons, 1996
- [13] A. Burns and A.J. Wellings, "Real-time Systems and Programming Languages: Ada 95, Real-Time Java and Real-time POSIX", Addison Wesley Longmain , 2001
- [14] L. Sha, el al, "Priority Inheritance Protocols: An Approach to Real-time Synchronization", IEEE Transactions on Computers, 39(9):1175–1185, September 1990
- [15] X. Leroy, "Java Bytecode Verification: An Overview", Proceedings of CAV'01, number 2102 in LNCS, pages 265–285. Springer
- [16] T.J. Wilkinson and Associates, "Kaffe: A Free Virtual Machine to Run Java Code", Technical report, <http://www.kaffe.org>
- [17] H. Boehm et al, "Tree Benchmark for Memory Management", http://www.hpl.hp.com/personal/hans_Boehm/gc/gc_bench.html, accessed April 2003

Aspect-Orientation of Control Application Code for Distributed Automation Systems: The TORERO Approach

Marcus Tangermann¹, Christian Schwab¹, Athanasios P. Kalogeras²,
Kai Lorentz¹, and Aggeliki S. Prayati³

¹University of Magdeburg, Center Distributed Systems@IAF,
Universitätsplatz 2, 39106 Magdeburg, Germany
{marcus.tangermann,christian.schwab,kai.lorentz}
@mb.uni-magdeburg.de

²Industrial Systems Institute, Building A, University Campus,
26500 Rion Patras, Greece
kalogeras@isi.gr

³University of Patras, Dept. of Electrical Engineering & Computer Technology,
26500 Rion Patras, Greece
prayati@ee.upatras.gr

Abstract. Besides the programming of the control application code of a distributed automation system (DAS) and the functional allocation of the control application code to the specific devices that the DAS consists of, an important point is the implementation of communication related code to map to the underlying communication platform and to realise the collaboration of the distributed control application code. Due to the real-timeliness and even time-criticality of DAS applications, the correct handling of real-time constraints has to be considered. This paper presents an aspect-oriented approach for weaving the communication related code into the distributed control application code by means of AspectJ [AspJ03], an extension for aspect-oriented programming with Java, and gives a comparison to other approaches. In addition to the description of the weaving process and the presentation of a demonstrator example, considerations about the usability of this concept and some restrictions will be stated. This work is part of the research project TORERO, which is funded by the European Commission under the IST program.

1 Introduction

The current situation in automation systems is characterised by a strong trend towards distributed systems based on intelligent devices [3, 16, 20, 21]. From the software point of view, a major milestone for this is the international standard IEC 61499 [12, 15]. Relevant to this, several proposals created by different organisations and companies exist, specifying a future way of building industrial automation applications [1, 11, 17]. All these approaches have the common modelling paradigm of programming distributed applications by utilizing and interconnecting elementary function blocks residing on DAS devices via a “system editor”. However, a wide variety of problem solutions such as the specific ways

- to design distributed control applications independent to the underlying resources (e.g. communication protocol or device hardware characteristics like processing power and memory),
- to perform optimal allocation,
- to implement communication related code,
- to deploy the control application code,
- to integrate the Internet into the automation system down to the device (e.g. for maintenance purposes), and thus
- the support of the total life cycle of the automation system

are still under development.

In this respect, the research project TORERO (Total life cycle web-integrated control) - funded within the IST initiative of the European Commission - aims at specifying both an Integrated Development Environment (TORERO IDE) and an architectural platform on specific devices (TORERO Devices) which together allow for the distribution of control applications based on event driven Functions Blocks (FBs) related to IEC 61499 standard [22]. The FBs can be accessed either locally or remotely over the network and the appropriate automation protocol. The development of the control application code (FBs) is independent of the afterwards distribution of this code.

Some of the aspects of distributed systems include component interaction, synchronisation, remote invocation, parameter transfer strategies and failure handling. Some aspects follow structures which naturally crosscut generalised procedures such as control flow or data flow. For example, synchronisation, real-time constraints, and object interaction follow control flow and parameter-transfer strategies in distributed systems. The work presented in this paper focuses on the usage of aspect-oriented languages for weaving communication related code into the distributed application control code.

The control application code of the DAS is programmed in Java regarding the Real-Time Specification for Java [18, 19] with support by the TORERO IDE. The distribution of control applications will be realised in two steps in TORERO. Step 1 is associated with the semi-automated allocation of the control application code to the single devices, while step 2 covers the implementation of the communication related code necessary for the proper functioning of the distributed control application. Once the control application code is allocated to the single devices, the implementation of communication related code into the allocated control application code will be done automatically by the TORERO IDE by means of the aspect-oriented language AspectJ [4]. This approach will be described in the following sections.

2 Aspect-Oriented Approach in Automation

Aspect-oriented programming (AOP) is a software development paradigm that aims at attaining a higher level of separation of concerns in both functional and non-functional matters by introducing a new modular unit, called aspect. The idea of AOP is to separate the component code from the aspect code. Thus, it will be possible to develop the functional program at first (control application code), and then add specific aspects (e.g. communication related code covering an automation protocol

such as Modbus/TCP [13, 14]) at a later point in the development phase of the project. This leads to a resource (hardware) independent programming of the control application code and to an increasing possibility of reusing software components. The aspect code can consist of several aspect programs, each of which implements a specific aspect (e.g. different automation protocols used in the system). Crosscutting lies at the heart of aspects. Modular units of decomposition are organised into clear hierarchies, whereas aspects crosscut such hierarchies. Join points act as the location where the aspect code is inserted into the component code, where the crosscut is. In general, we can distinguish between three types of join points between aspects:

- methods, attributes. For example, an aspect could refer to the definition of a method and state that calls to this method should be logged to a file. Thus, by referring to the definition, we affect all calls to the method.
- Qualified “by name” references. Sometimes we do not want to log all calls to the method M1 but only those made within M2. In this case, the aspect would make a reference to M1 qualified with M2. Thus, qualified references allow to refer to some points of a language construct.
- References to patterns. Traversal strategies are an example of pattern-based coupling. A behavior specification may refer only to certain regions in a class graph using traversal strategies, i.e. patterns.

The Aspect Weaver is the core component of AOP, which takes the aspect code and the component code, finds join points and weaves them all together to form a single entity [2], this approach is presented in Fig. 1.

According to the time the code is combined there are two different options for an aspect weaver:

- static: The code is combined during compilation time and may not be changed at runtime.
- dynamic: The code is combined at runtime. This is the most flexible variant of AOP since it uses a minimal coupling between aspect code and component code.

To implement AOP in Java, AspectJ was created, consisting of a special aspect language and a static code weaver. This project is supported by different tools, e.g. by plug-ins for the Eclipse development tool [7].

3 Weaving of Communication Related Code in IEC 61499 FBs

Providing support for an aspect involves two things:

- implementing abstractions for expressing the aspect,
- implementing weaving for composing the aspect code with the primary code and the code for other aspects.

There exist three possible ways to implement the aspects:

- Encode the aspect support as a conventional library,
- Design a separate language for the aspect,
- Design a language extension for the aspect.

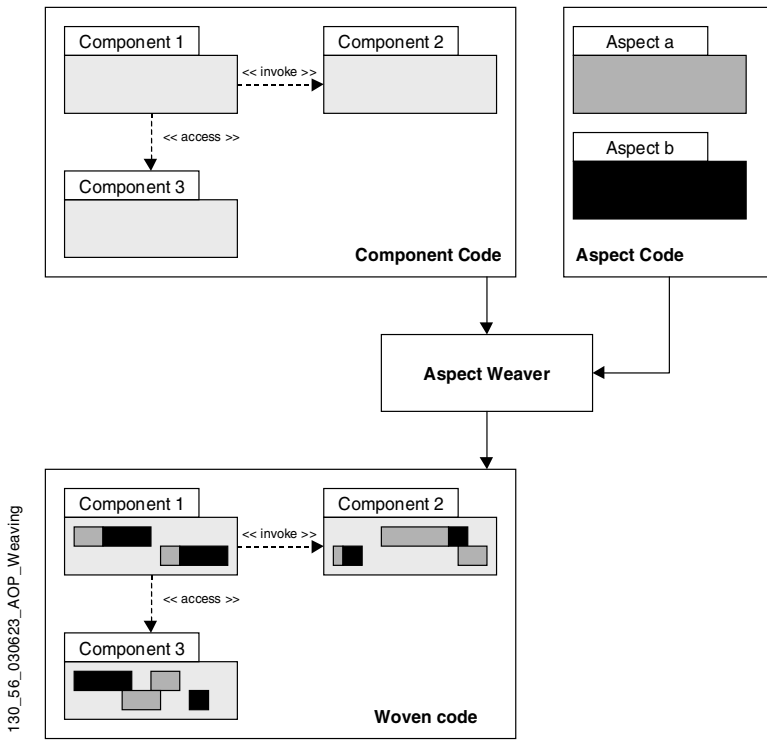


Fig. 1. Weaving of aspect code into component code [10]

Among the three possibilities for capturing aspects, modular language extensions have a number of advantages and thus are more preferable than a fixed set of separate languages. Language extensions are more scalable. Plug and unplug is possible, a feature particularly useful, when, during system development, more aspects need to be addressed. Moreover, language extensions allow the reuse of compiler infrastructure and language implementation. Also, one extension can work with many others.

Using AOP to implement the communication related code into the control application code gives the following benefits:

- Generic approach for the communication interface,
- Possibility to use different communication protocols without any changes in the control application code,
- Integration of local and remote access without adaptation of the control application code,
- Open for extensions, e.g. the integration of a new communication protocol,
- Increase in the level of interoperability and interworkability of a DAS with reference to the communication infrastructure.

The following example illustrates the basic elements of AOP and especially of AspectJ by means of implementing parts of an IEC 61499 FB [6]. It develops two

variants of an aspect using AspectJ by defining the necessary elements and finally combining them to an aspect.

An IEC 61499 FB follows an event driven approach and consists of event inputs and event outputs that trigger algorithms by means of an Execution Control Chart (ECC). A FB utilizes its appropriate data inputs, executes its control algorithms, and finally writes its data outputs (Fig. 2) [5].

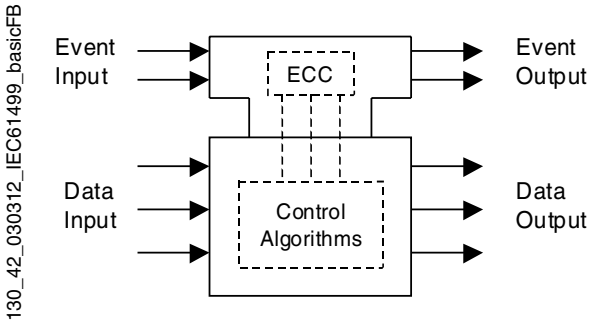


Fig. 2. IEC 61499 FB

As mentioned in the introduction of this paper those FBs which comprise a control application may be either local or distributed over the network. In Fig. 3, these two different options are depicted. On the first hand, the function block FB 1 resides on Device 1 and triggers locally FB 2 and FB 3 also residing on the same device. On the other hand, the function block FB 1 residing on Device 1 triggers remotely FB 2 and FB 3 residing on Device 2, by means of the communication infrastructure, which could for instance be a network like Ethernet combined with an appropriate automation protocol, e.g. Modbus/TCP.

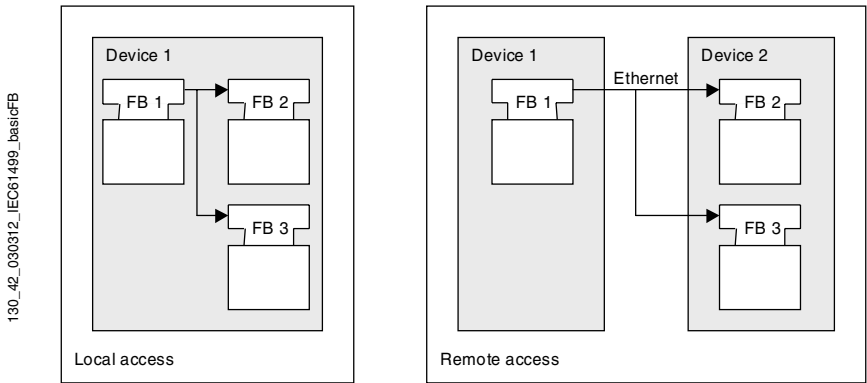


Fig. 3. Distribution of FBs

As a result, the communication between the FBs may vary a lot, ranging from local procedure calls to the utilisation of services of the industrial communication protocol infrastructure. In the former case the FB has to call the appropriate receive method of the local FB while in the latter case the FB has to initialise the automation protocol such as Modbus/TCP or EtherNet/IP [8]. Thus, it is evident that the communication itself does not present an elementary part of the FB, it is rather “something that can be said about it”, e.g. “the FB communicates over the network” or “the FB calls local methods” illustrating the fact that communication is actually an aspect of the FB.

A simplified version of the class `FunctionBlock` (component code) is shown in the following listing:

```
//Listing 1
import javax.realtime.* ;
...
public class FunctionBlock extends RealtimeThread{
    ...
    public FunctionBlock(String name){
        // Initialisation of the function block
        ...
    }
    public void run(){
        // Execution Control Chart
        // Control Application Code
        ...
    }
    public void send_event(byte[] data){ }
        // Do nothing, build by aspect code
    public void receive_data(byte[] data){
        // Receive the data
        ...
    }
}
```

As a consequence of the above definition, the FB class will be programmed independently from the communication and hence independently from the underlying hardware. Instead the communication process will be considered as an aspect containing a pointcut, a description regarding which join points (well-defined points in the program flow or component code) need to be altered. In the presented example and regarding communication, we need to add code for the `send_event(byte[])` method of the FB to the aspect.

The declaration for this pointcut in AspectJ as shown in Listing 2 contains the following elements:

- The name of the pointcut: `send_event`, and
- Two parameters: `FunctionBlock f` and `byte[] data`.

```
//Listing 2
pointcut send_event(FunctionBlock f, byte[] data):
target(f)      && args(data)
               && call(public void send_event(byte[]));
```

After the colon it is defined which joinpoints this pointcut is responsible for:

- This pointcut is restricted to `FunctionBlock` objects (`target(f)`).
- This pointcut is responsible for calls to `send_event` with a byte array as parameter.

The rule `args(data)` tells the weaver that `data` is defined as an argument for later use, in this case it act as the parameter for `send_event`. After describing where the code has to be added, the next step is to program the code to add (aspect code), called an advice. An example for an aspect for a remote access is shown in Listing 3.

```
//Listing 3
before(FunctionBlock f, byte[] data):
send_event(f,data) {
    try{
        Socket s = new Socket
            (InetAddress.getByName("192.168.22.101"),
10000);
        System.out.println("Socket open");
        // Send data according to the actual protocol
        ...
    } catch(Exception e){
        System.out.println("An exception occured: " +
e.getClass()+"
"+      e.getMessage());
    }
}
```

The directive `before` tells the compiler to insert this code before the pointcut occurs in the original program. As a pointcut `send_event` is used which was defined in Listing 2. This declaration shows the usage of the two parameters declared in the pointcut, the `FunctionBlock f` and `byte[] data`, treated as parameters of this advice.

Finally the aspect code is completed with the appropriate import statements for the network code (`java.net.*` and `java.io.*`) and the declaration of the aspect name (Listing 4).

```
//Listing 4
import java.net.* ;
import java.io.* ;

aspect Protocol {
    // Pointcut code, see Listing 2
    // Advise code, see Listing 3
}
```

Now, the aspect code can be inserted accordingly by the aspect weaver, and the networking functionality is available to the FBs.

As mentioned above it may also be possible that the FBs communicate with other local FBs. In this case, it is not sufficient to simply add code to the method `send_event`, but it is also necessary to have a mechanism to get a reference to the FB,

to which data shall be sent. For simplicity, a `Hashtable` will be used in this example where a unique name of the FB is stored together with a reference to the FB-object.

The registration of a FB shall be processed automatically, so the best joinpoint would be the constructor of the class `FunctionBlock`. The according pointcut is depicted in Listing 5.

```
//Listing 5
pointcut register(FunctionBlock f, String name):
    target(f)
    && args(name)
    && execution(FunctionBlock.new(String));
```

The first part of the declaration is equal to the pointcut of the first example, the difference lies in the usage of the keyword `execution` instead of `call`. Every time the constructor of `FunctionBlock` is executed with a parameter `String` this pointcut is used. By the usage of new `AspectJ` refers to a constructor.

As the next step the appropriate advice is programmed as depicted in Listing 6.

```
//Listing 6
protected Hashtable blocks = new Hashtable() ;
...
after(FunctionBlock f, String name):register(f,name) {
    blocks.put(name,f);
    ...
}
```

After the execution of the constructor the instance of `FunctionBlock f` is put to the hashtable with the key name. With this advice it is secured that every FB can be referenced by its name. In the next step a new pointcut for `send_event` is defined which is equal to Listing 2. The final step represents the programming of a new advice for the `send_event` method which is shown in Listing 7.

```
//Listing 7
before(FunctionBlock f, byte[] data):
send_event(f,data) {
    FunctionBlock f2 = (FunctionBlock)
blocks.get("TestBlock2") ;
    f2.receive_data(data) ;
}
```

In this advice the appropriate `FunctionBlock` instance is retrieved from the hashtable and the `receive_data` method is called.

4 Comparison to Other Approaches

During research of the object oriented system design and programming a wide variety of methods that seem to be suitable for the solution of the above described problem has been developed. Within this section a comparison with two of these methods will

be given and it will be outlined why AOP is the preferable approach for adding the communication code to IEC61499 function blocks.

When discussing remote calls of functions on different network nodes using Java, the Remote Method Invocation (RMI) first comes in mind. Comparable to the common Remote Procedure Calls within Unix/Linux and Windows it addresses the problem of distributed computing based on IP networks. Facing the the automation market the system designer is confronted with a wide variety of *automation* protocols that he has to bear in mind during the design process. Other communication procedures such as RMI are not accepted by the market, at least at the moment.

One of the advantages of the object oriented design is the management of knowledge using design patterns. It is a common way of reusing approved solutions for different kinds of problems reducing development time and cost based on the know-how of experienced developers. For example the factory pattern [9] tries to eliminate the need to bind application-specific code into your application. Therefore a creator class decides which concrete implementation has to be loaded at runtime, the communication with the calling application is realized via an interface. At a first glance this seems to be another solution to the problem of adding communication code to the function blocks used within the TORERO project. But in detail some problems occur that can be avoided using AOP:

- Due to the usage of an interface class all methods need the same set of parameters. Variable parameters could be emulated using `Vector` objects containing `Object` instances as parameters, but this could lead to problems using areas without heap (e.g. `NoHeapRealtimeThreads`) and also result in performance loss due to the excessive use of typecasting
- The factory pattern method is only applicable if the functionalities of all implemented protocols are equal in the sense that they have an equal set of calls. In the scenario described in the last section it is outlined that the local distribution of function blocks leads to different function calls at different points in the control flow than the usage of network communication.

As the consequence the application of the factory pattern leads to manual changes of code by the designer/programmer depending on the used communication method that can be avoided using AOP.

5 Summary and Outlook

The example presented in this paper has shown how aspect oriented programming supports the development of function blocks in TORERO. The decision to use AOP is based on the facts that

- it is open to extensions,
- it provides a generic approach for the communication interface,
- it allows the use of different communication protocols without any changes in the control application code, as well as the integration of local and remote access without adaptation of the control application code.

Moreover, by using AOP and by increasing the abstraction and crosscutting levels of programming languages, it is possible to eliminate some of the manual, algorithmic

work required by the programmer. By moving some of the manual work into the aspect weavers, it is easier to concentrate on the more creative parts of software development and enable the construction of even more complex systems. These special benefits of the application of AOP bring a higher degree of flexibility into the programming of IEC 61499 function blocks. Only one basic FB implementation is sufficient for all kinds of usage, independently from the distribution (local or network) or the used protocol.

The current implementation of AspectJ is based on the insertion of specific classes into the function code, which raises some issues regarding the RTSJ that need attention. A first issue occurs when AspectJ is combined with NoHeapRealtime Threads. Since developed for the common garbage collection, it is not designed to avoid garbage, which is essential when working with threads without heap.

Another issue that has to be examined further is the overall impact of the inserted code on the real-time performance. The implementation of the code weaver (in this case AOP) may influence the program flow, so an analysis of the specific implementation of AspectJ is necessary.

As a conclusion it can be stated that the application of AOP offers a lot of benefits for the usage in embedded and real-time systems, a proof of concept is shown in the TORERO project. Special issues regarding RTSJ will be further examined and it is expected that they will be solved in the near future.

References

1. N.N.: Offene Plattformen für den Mittelstand (Transl. Open Platforms for medium-sized and small enterprises) in IEE Automatisierung + Datentechnik, Hühlig GmbH & Co. KG, Heidelberg, March 2003.
2. Gregor KICZALES, John LAMPING, Anurag MENDHEKAR, Chris MAEDA, Cristina Videira LOPES, Jean-Marc LOINGTIER, John IRWIN: Aspect-Oriented Programming in Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag, June 1997.
3. ARC Advisory Group: PLCs: Is there a future? ARC Strategies, Dedham: ARC, 2001.
4. AspectJ, <http://www.eclipse.org/aspectj/>, Version 1.1.0, June 2003.
5. James H. CHRISTENSEN: Basic Concepts of IEC 61499, Fachtagung 2000 Verteilte Automatisierung – Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung (Tagungsband), Univ. of Magdeburg, Germany, 2000.
6. The TORERO consortium: Deliverable 2.1 Integrative design and development of web-enabled control system design methodology (internal draft), 2003.
7. The Eclipse Project: www.eclipse.org.
8. EtherNet/IP Specification, Release 1.0. ODVA, June 2001. (www.ethernetip.de and www.odva.org)
9. Erich Gamma, Richard Helm, Ralph Johnson: Design Patterns – Elements of reusable object-oriented Software, Addison-Wesley Professional, 1997.
10. A. GAL, W. SCHRÖDER-PREIKSCHAT, and O. SPINCZYK: On Aspect-Oriented Distributed Real-time Dependable Systems; University of Magdeburg, Germany, 2001.
11. IDA-Group: Interface for Distributed Automation – Architecture Description and Specification, Revision 1.1, Blomberg, November 2002.
12. IEC-PAS 61499-1, Function Blocks for Industrial Process Measurement and Control Systems, Edition 1.0, 2000-09, VDI Verlag, Düsseldorf, 2000.

13. Modbus User Community: Modbus Messaging on TCP/IP, Implementation Guide, May 2002. (www.modbus.org)
14. Modbus Application Protocol, V1.1, December 2002. (www.modbus.org)
15. P. NEUMANN: Aktuelle Triebkräfte der Entwicklung von Automatisierungssystemen, Fachtagung 2000 Verteilte Automatisierung – Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung (Tagungsband), Univ. of Magdeburg, Germany, 2000.
16. Profibus Nutzer Organisation: ProfiNet – More than just Ethernet, Karlsruhe, PNO, 2001.
17. PNO: PROFInet – Architecture Description and Specification, Version V2.0, Karlsruhe, January 2003.
18. Peter C DIBBLE: Real-Time Java Platform Programming, Prentice Hall Computer Books, 2002.
19. Real-Time Specification for Java (First Public Release), Addison-Wesley, 2001.
20. C. SCHWAB, A. P. KALOGERAS, K. LORENTZ, et.al.: Next Generation Integrated Development of Automation Control Code in TORERO, IEEE International Symposium on Industrial Electronics – ISIE, Rio de Janeiro, Brazil, June 2003.
21. M. TANGERMANN: TORERO – Total Life Cycle web-integrated Control, Presentation at OMG Conference, Working Group Real-Time and Embedded Specialised Systems, Paris, France, June 2003.
22. The TORERO consortium, www.torero-project.com.

JOP: A Java Optimized Processor

Martin Schoeberl

JOP.design, Strausseng. 2-10/2/55, A-1050 Vienna, Austria
martin@jopdesign.com

Abstract. Java is still not a common language for embedded systems. It possesses language features, like thread support, that can improve embedded system development, but common implementations as interpreter or just-in-time compiler are not practical. JOP is a hardware implementation of the Java Virtual Machine with focus on real-time applications. This paper describes the architecture of JOP and proposes a simple real-time extension of Java for JOP. First application in an industrial system showed that JOP is one way to use Java in the embedded world.

1 Introduction

Current software design practice for embedded systems is still archaic compared to software development for desktop systems. C and even Assembler is used on top of a small RTOS. The variety of embedded operating systems is large and this fragmentation of the market leads to high cost. Java [1] can be a way out of this dilemma and possess language features not found in C:

- Object-oriented
- Memory management with a garbage collector
- Implicit memory protection
- Threads

Memory management and threads are (besides device drivers) the main components of embedded operating systems. Finding these features in the language embedded systems can be programmed in Java without the need of an operating system.

Java on desktop systems comes with a large library. However, if Java is stripped down to the core components it has a very small memory footprint. With careful programming (like using only immortal memory as in [2]) the garbage collector can be avoided. Without a GC, Java can be used even in hard real-time systems.

The definition of the language includes also the definition of the binary and the Java Virtual Machine (JVM) [3] to execute these programs. The JVM is a stack machine and can be implemented in several ways:

Interpreter: A simple solution with low memory requirements, but lacks in performance.

Just-in-Time Compilation: Got very popular on desktop systems, but has two main disadvantages in embedded systems: A compiler is necessary on the target and due to compilation during runtime execution times are not predictable.

Batch Compilation: Java can be compiled in advance to the native instruction set of the target. Dynamic loading of classes is no longer possible (not a main concern in embedded systems).

Hardware Implementation: A Java Processor with JVM bytecodes as native instruction set.

JOP is a hardware implementation of the JVM targeted for small embedded systems with real-time constraints. It shall help to increase the acceptance of Java for those systems.

JOP is implemented as a soft core in an FPGA (Field Programmable Gate Array). Using an FPGA as processor for embedded systems is uncommon due to high cost compared to a micro controller. However, if the core is small enough, unused FPGA resources can be used to implement periphery in the FPGA resulting in a lower chip count and hence lower overall cost.

The main features of JOP are summarized below:

- Fast execution of Java bytecodes without JIT-Compiler.
- Predictable execution time of Java bytecodes.
- Small core that fits in a low cost FPGA.
- Configurable resource usage through HW/SW co-design.
- Flexibility for embedded systems through FPGA implementation.

2 Architecture

Every design is influenced by the available tools. In architecture, the constraints will be different whether we use wood, stone or steel. The same is true for CPU architecture. The first and primary implementation of JOP is in an FPGA.

2.1 FPGA Implementation

An FPGA has two basic building blocks: logic elements and memory. A logic element (LE) consists of a 4-bit LUT (Look Up Table) and a flip-flop. Memory blocks (ESB) are usually small (e.g. 0.5 KB) with independent read and write ports of configurable size. With these constraints, a stack machine is an attractive architecture in an FPGA:

- The stack can be implemented in internal memory.
- A register file in a RISC CPU needs two read ports and one write port for single cycle instructions. A stack needs only one read and one write port (common in current FPGAs).
- Instruction set is simpler and can be reduced to 8 bit.
- No data forwarding is necessary.

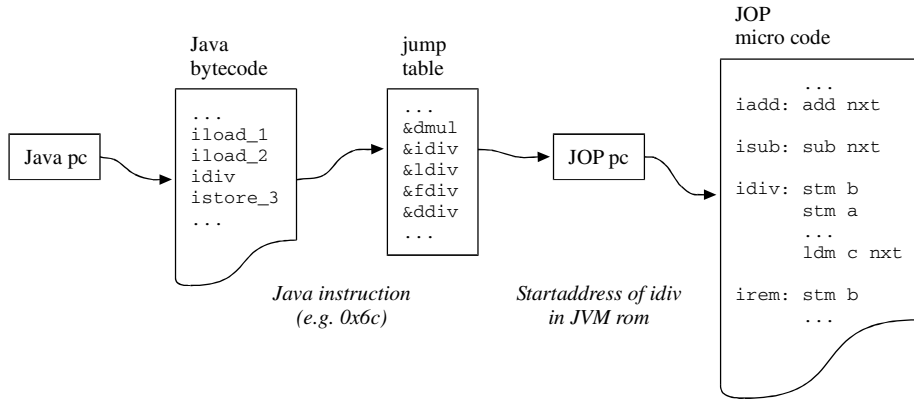


Fig. 1. Data flow from the Java program counter to JOP micro code. The fetched bytecode is used as an index into the jump table. The jump table contains the start addresses of the JVM implementation in micro code. This address is loaded into the JOP program counter for every executed bytecode.

2.2 Micro Code

There is a great variation in complexity of Java bytecodes, the instructions of the JVM. There are simple instructions like arithmetic and logic operations on the stack. However, the semantics of instructions like *new* or *invokestatic* can result in class loading and verification. Because of this variation, not every JVM instruction can be implemented in hardware. One common solution, used in Suns picoJava-II [5], is to execute a subset of the bytecode native and trap on the more complex ones. This solution has a constant overhead for the software trap.

The approach to this problem in JOP is different. JOP has its own instruction set (the so called micro code). Some bytecodes have a 1 to 1 mapping to JOP instructions, for the more complex a sequence of JOP instructions is necessary. Every bytecode is translated to an address in the micro code that implements the JVM. Fig. 1 shows an example of this indirection.

If the bytecode has an equivalent JOP instruction, it is executed in one cycle and the next bytecode is translated. For more complex bytecodes JOP just continues to execute micro code in the following cycles. The end of this sequence is coded in the instruction (as the *nxt* bit). This translation needs an extra pipeline stage but has zero overheads for complex JVM instructions.

```

dup:      dup nxt      // 1 to 1 mapping

// a and b are scratch variables for the JVM code.
dup_x1:   stm a        // save TOS
          stm b        // and TOS-1
          ldm a        // duplicate former TOS
          ldm b        // restore TOS-1
          ldm a nxt    // restore TOS and fetch next bytecode
  
```

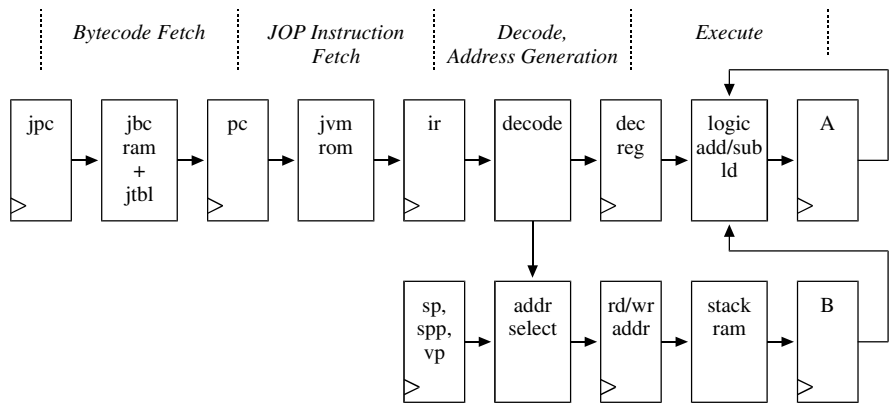


Fig. 2. Pipeline of JOP

This example shows the implementation of a single cycle bytecode (*dup*) and an unusual bytecode (*dup_x1*) as a sequence of JOP instructions that take 5 cycles to execute.

2.3 Pipeline Overview

The stack architecture allows a short pipeline resulting in short branch delays. Fig. 2 shows an overview of the pipeline. Three stages form the core of JOP, executing JOP instructions. An additional stage in the front of the core pipeline translates bytecodes to addresses in micro code. Every JOP instruction takes one cycle. Conditional branches have an implicit delay of two cycles. This branch delay can be filled with instructions or *nop*.

2.4 Java Bytecode Fetch

The first pipeline stage can be seen in Fig. 3. All bytecodes are fetched from internal memory (*bytecode ram*). This memory, the instruction cache, is filled on function call and return. Every byte is mapped through *jtbi* to an address for the micro code rom (*jpaddr*). It is also stored in a register for later use as operand. Since *jpc* is also used to read operands, the program counter is stored in *jpcbr* during an instruction fetch. *jinstr* is used to decode the type of a branch and *jpcbr* to calculate the target address.

2.5 JOP Instruction Fetch

Fig. 4 shows the second pipeline stage. JOP micro code that implements the JVM is stored in the memory labeled *jvm rom*. The program counter *pc* is incremented during normal execution. If the instruction is labeled with *nxt* a new bytecode is requested from the first stage and *pc* is loaded with *jpaddr*. *jpaddr* is the starting address for the implementation of that bytecode. This label and the one for a bytecode operand load (*opd*) are stored in *bc-fetch*.

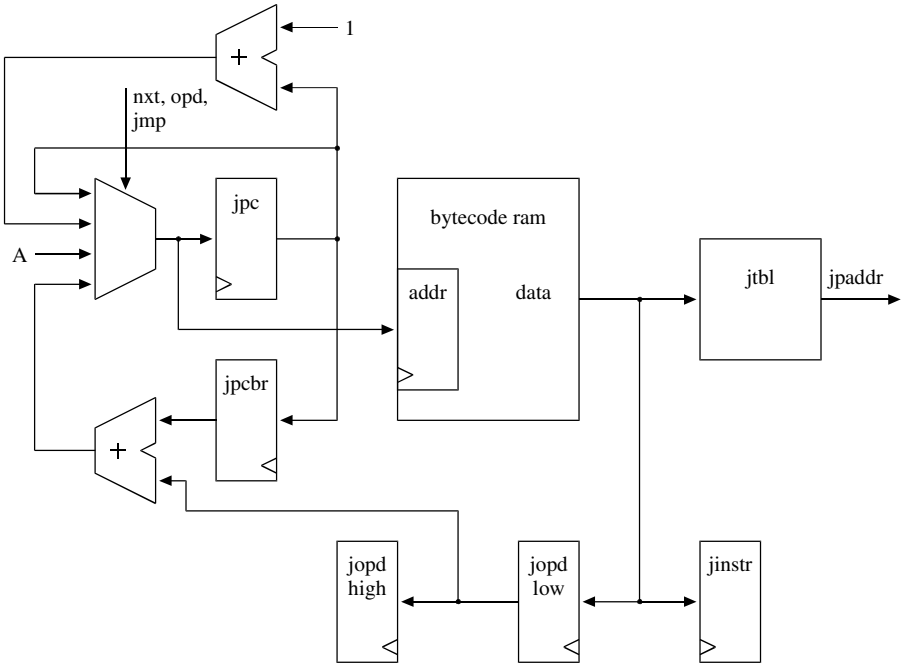


Fig. 3. Java bytecode fetch

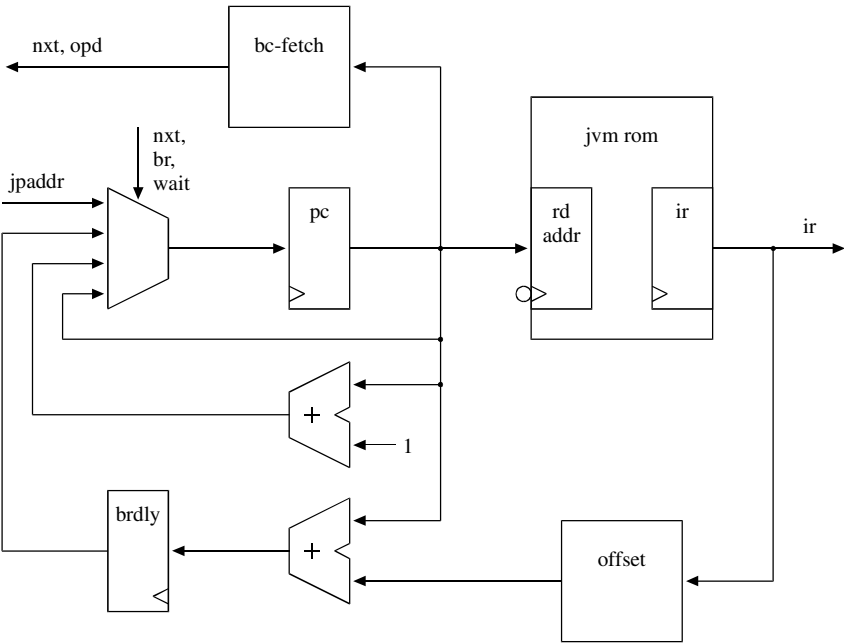


Fig. 4. JOP instruction fetch

brdly holds the target for a taken conditional branch. Many branch destinations share the same offset. A table (*offset*) is used to store these relative offsets. This indirection makes it possible to use only five bits in the instruction coding for branch targets and allow larger offsets. The three tables *bc-fetch*, *offset* and *jtbl* (from the bytecode fetch stage) are generated during assembly of the JVM code. The outputs are VHDL files. For an implementation in an FPGA it is no problem to recompile the design after changing the JVM implementation. For an ASIC with loadable JVM a different solution is necessary.

Current FPGAs don't allow asynchronous memory access. They force us to use the registers in the memory blocks. However, the output of these registers is not accessible. To avoid an additional pipeline stage just for a register-register move the read address register is clocked on the negative edge.

2.6 Decode and Address Generation

The third pipeline stage shown in Fig. 5 provides two functions. JOP instructions are decoded for the execution stage and addresses for read and write accesses of the stack ram are generated.

Instructions of a stack machine can be categorized with respect to stack manipulation in *pop* or *push*. This allows us to generate the addresses for fill or spill of TOS-1 for the *following* instruction during the decode stage, saving one extra pipeline stage.

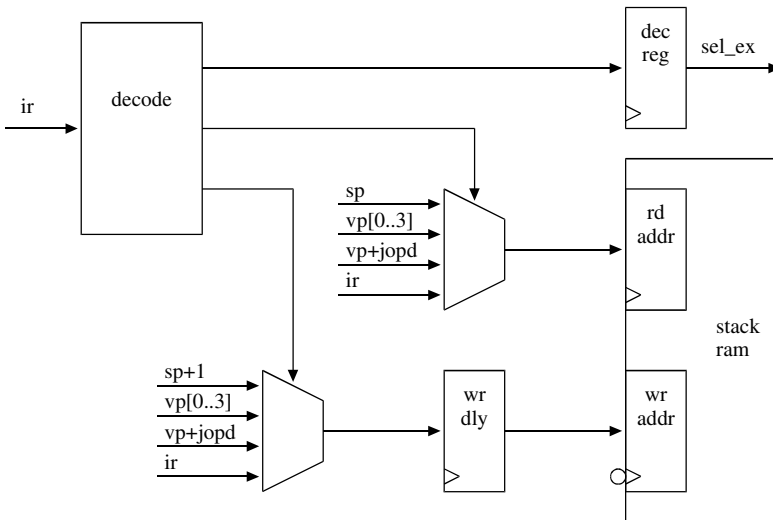


Fig. 5. Decode and address generation

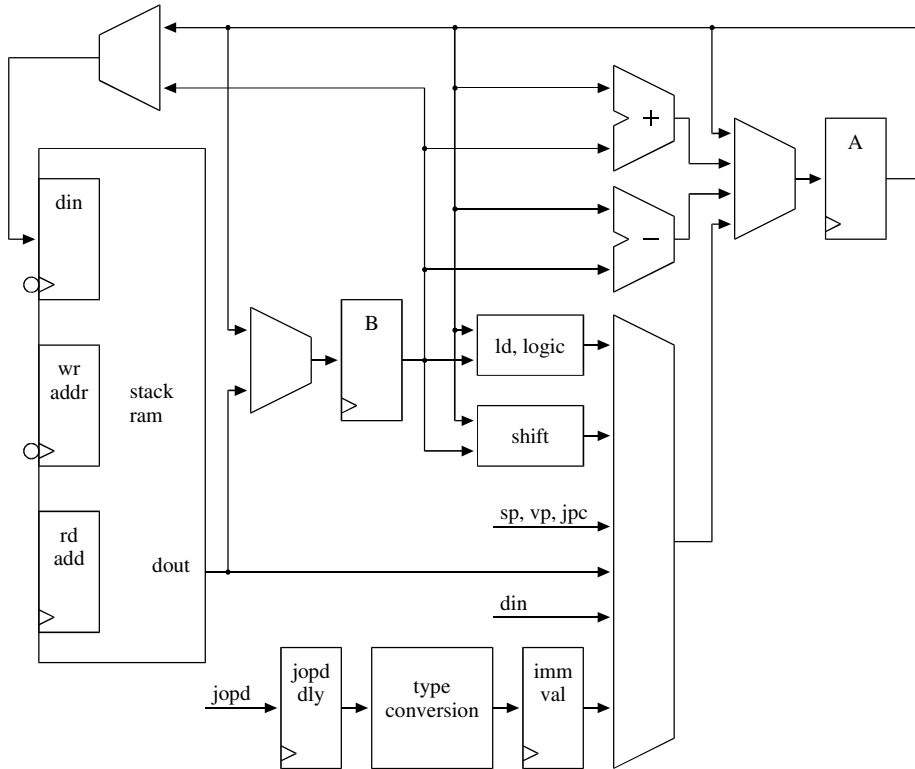


Fig. 6. Execution stage

2.7 Execute

As can be seen in Fig. 6 TOS and TOS-1 are implemented as register *A* and *B*. Every arithmetic/logical operation is performed with *A* and *B* as source and *A* as destination. All load operations (local variables, internal register, external memory and periphery) result in the value loaded in *A*. Therefore no write back pipeline stage is necessary. *A* is also the source for store operations. Register *B* is never accessed directly. It is read as implicit operand or for stack spill on push instructions and written during stack spill and fill.

3 HW/SW Co-design

Using a hardware description language and loading the design in an FPGA, the traditional strict border between hardware and software gets blurred. Is configuring an FPGA not more like loading a program for execution?

This looser distinction makes it possible to move functions easily between hardware and software resulting in a highly configurable design. If speed is an issue, more functions are realized in hardware. If cost is the primary concern these functions

are moved to software and a smaller FPGA can be used. Let us examine these possibilities on a relatively expensive function: multiplication. In Java bytecode *imul* performs a 32 bit signed multiplication with a 32 bit result. There are no exceptions on overflow.

Since single cycle multiplications for 32 bits are far beyond the possibilities of current FPGAs, we can implement *imul* with a sequential booth multiplier in VHDL. Three JOP instructions are used to access this function: *stopa* stores the first operand and *stopb* stores the second operand and starts the sequential multiplier. After 33 cycles, the result is loaded with *ldmul*.

If we run out of resources in the FPGA, we can move the function to micro code. The implementation of *imul* needs 73 JOP instructions and has an almost constant execution time.

JOP micro code is stored in an embedded memory block of the FPGA. This is also a resource of the FPGA. We can move the code to external memory by implementing *imul* in Java bytecode. Bytecodes not implemented in micro code result in a static method call from a special class (`com.jopdesign.sys.JVM`). The class has prototypes for every bytecode ordered by the bytecode value. This allows us to find the right method by indexing the method table with the value of the bytecode. The additional overhead for this implementation is a call and return with the cache refills.

Table 1 lists the resource usage and execution time for the three implementations. Executions time is measured with both operands negative, the worst-case execution time for the software implementations. The implementation in Java loads bytecodes from a slow memory interface (8 bit, 3 cycle per byte) and execution time depends on the caching policy.

Table 1. Different implementations of *imul*

	Hardware [LE]	Micro code [Byte]	Time [Cycle]
VHDL	300	12	37
Micro code	0	73	750
Java	0	0	~2300

Only a few lines of code have to be changed to select one of the three implementations. The showed principle can also be applied to other expensive bytecodes like: *idiv*, *ishr*, *iushr* and *ishl*. As a result, the resource usage of JOP is highly configurable and can be selected for every application. The possibility to call Java methods from micro code also allows us to code part of the JVM (like thread scheduling) in Java.

4 Real-Time Predictability

In real-time systems, especially hard real-time systems, meeting time constraints is of the same importance as functional correctness. One way to prove that all timing requirements are met is to calculate WCET (Worst-Case Execution Time) of all tasks.

These WCET values are the input for schedulability analysis. High-level WCET analysis, analyzing possible program flows, is a well-established research area [6]. A

framework for portable Java bytecode WCET analysis can be found in [7]. At the low-level analysis execution time of bytecodes are derived from a specific VM model. The accuracy of the model has a major impact on the tightness of the WCET. Caches and pipeline effects are hard to model and can lead to an overestimation. Pipeline effects of common pairs of bytecodes are modeled in [8] to get tighter WCET.

These problems can be avoided with a well-known timing behavior for each bytecode and the cache. In JOP the execution time of all bytecodes is known cycle accurate. Most instructions have a constant execution time. Although JOP is full pipelined, resulting in some single cycle bytecodes, there are no timing dependencies between successive bytecodes. Even the conditional branch bytecodes have constant 4-cycle execution time whether the branch is taken or not.

4.1 Time Predictable Caches

The memory areas of the JVM can be classified as follows:

- Class description with method table and constant pool
- Code area
- Stack
- Heap for objects

We can decide which memory areas are cached. The two most frequent accessed areas are stack and code. The stack is implemented as internal memory in JOP resulting in a single cycle cache with independent read and write ports.

The same is true for code memory. Every executed bytecode is read from an internal memory (i.e. instruction cache). However, there is a big difference between JOPs instruction cache and instruction caches found in typical processors: No cache misses are allowed! This allows absolute predictable modeling of the cache behavior.

When is this cache filled? Typical Java programs consist of short methods. There are no branches out of the method and all branches inside are relative. In JOP the full code of a method has to be loaded in the cache before execution. The cache is filled on calls and returns. This means that all cache fills are lumped together with a known execution time.

The full loaded method and the relative addressing inside a method results in a simpler cache. No tag memory and no address translation are necessary.

The remaining two memory areas are not cached in JOP. Currently only one method is cached resulting in a refill on every method return. This solution has a predictable execution time but the average performance is worse than a traditional instruction cache. Keeping more methods in the cache with an efficient and predictable replace policy is a topic for further research.

4.2 Simple Real-Time Extension for Java

Tasks or threads are vital program constructs in embedded programming. Since threads and synchronization are defined as part of the language, Java can greatly simplify concurrent programming. Java, as described in [1], defines a very loose behavior of threads and scheduling. E.g. the specification allows even low priority threads to preempt high priority threads. This prevents threads from starvation in

general purpose applications, but is not acceptable in real-time programming. To compensate for this under-specification, extensions to Java have been published. The Real-Time Specification for Java (RTSJ) [2] developed under the Sun Community Process addresses this problem.

RTSJ is complex to implement and applications developed with RTSJ are (due to some sophisticated features of the RTSJ) difficult to analyze. Different profiles with restrictions of the RTSJ have been suggested. In [9] a subset of the RTSJ for high-integrity application domain, with hard real-time constraints, is proposed. It is inspired by the Ravenscar profile for Ada [10] and the focus is on exact temporal predictability. Ravenscar-Java profile [11], based on previous mentioned work, restricts RTSJ even further. It claims to be compatible with RTSJ in the sense that programs written according to the profile are valid RTSJ programs. However, mandatory usages of new classes like `PeriodicThread` need an emulation layer to run on an RTSJ system. In this case, it is better to define completely new classes for a subset and provide the mapping to RTSJ. This leads to clearer distinction between the two definitions.

This real-time extension resembles the ideas from [9] and [11] but is not compatible with RTSJ. Its main purpose is to provide a framework for the development of JOP. If only a subset of RTSJ is implemented and allowed it is harder for programmers to find out what is available and what not. Use of different classes for a different specification is less error prone and restrictions can be enforced (e.g. setting thread priority only in the constructor of a real-time thread).

Application Structure. Following restrictions apply to the application:

- Initialization and mission phase.
- Fixed number of threads.
- Threads are created at initialization phase.
- All shared objects are allocated at initialization.

Threads. Three schedulable objects are defined: *RtThread* represents a periodic task. As usual task work is coded in `run()` which gets called on `missionStart()`. *HwEvent* represents an interrupt with a minimum inter-arrival time. If the hardware generates more interrupts, they get lost. A software event (*SwEvent*) is scheduled after a call of `fire()`.

```
public abstract class RtTask {
    public void enterMemory()
    public void exitMemory()
}

public class RtThread extends RtTask {

    public RtThread(int priority, int usPeriod)
    public RtThread(int priority, int usPeriod, Memory mem)
    public void run()
    public boolean waitForNextPeriod()
}
```

```

public class HwEvent extends RtTask {

    public HwEvent(int priority, int usMinTime, int number)
    public void handle()
}

public class SwEvent extends RtTask {

    public SwEvent(int priority, int usMinTime)
    public SwEvent(int priority, int usPeriod, Memory mem)
    public final void fire()
    public void handle()
}

```

Definition of the basic classes for the simple real-time extension.

Scheduling. The class `Scheduler` defines a preemptive fixed priority scheduler (with FIFO within priorities). Synchronized blocks are executed with priority ceiling protocol. The scheduler does not dispatch any `RtThread` until `startMission()` is called. Standard Java threads are scheduled during initialization phase, however usage is discouraged. The scheduler provides access methods to measured worst-case execution time of periodic work and the handler method. These values can be used during application development when no WCET analysis tool is available.

Memory. The profile does not support a garbage collector. All memory allocation have to be done in the initialization phase. For new objects during mission phase a scoped memory is provided. A scoped memory area is assigned to one `RtThread` on creation. It is not allowed to share a scoped memory between threads. No references from the heap to scoped memory are allowed. Scoped memory is explicit entered and left with calls from the application logic. As suggested in [12] memory areas are cleared on creation and when leaving the scope (call of `exitMemory()`) leading to a memory area with constant allocation time.

An Example. Following code shows the principle coding of a worker thread, creation of two real-time threads and an event handler:

```

public class Worker extends RtThread {

    private SwEvent event;

    public Worker(int p, int t, SwEvent ev) {
        super(p, t, new Memory(10000));
        event = ev;
        init();
    }

    private void init() {
        // All initialization has to be placed here
    }
}

```

```

public void run() {

    for (;;) {
        work();           // do some work
        event.fire();     // and fire an event
        enterMemory();    // do some work in scoped memory
        workWithMem();
        exitMemory();
        if (!waitForNextPeriod()) {
            missedDeadline();
        }
    }
}

// Application initializing:
// Create an event handler and worker threads with
// priorities according to their periods
Handler h = new Handler(RtThread.MAX_PRIORITY, 1000);
FastW fw = new FastW(RtThread.MAX_PRIORITY-1, 2000);
Worker w = new Worker(RtThread.MAX_PRIORITY-2, 10000, h);

// Change to mission phase for all
// periodic threads and event handlers
Scheduler.startMission();

// Do some non real-time work
for (;;) {
    watchdogBlink();
    Thread.sleep(500);
}

```

5 Results

Table 2 compares the resource usage of different soft-core processors:

- Nios: [13] Alteras configurable load/store RISC processor.
- SPEAR: [14] Scalable Processor for Embedded Applications in Real-time Environments with 16-bit instruction set and 3-stage pipeline.
- Lightfoot: [15] Xilinx Java processor core, stack-based, 3-stage pipeline.
- JOP: with multiplier, single cycle shift, 8 bit memory interface, UART and timer.

Table 3 shows the result from a small benchmark. JOP runs at 24 MHz for this test and is compared with JVM version 1.1 on a Compaq notebook with Intel 486SX25. The 486SX has 1.2M transistors i.e. 300k gates. The FPGA where JOP is implemented for this comparison is (according to Altera) equivalent of 30k gates.

Table 2. Different FPGA soft cores

Processor	LEs	ESB	Data Path
Nios	1700	2.5 KB	32-bit
SPEAR	1700	8 KB	16-bit
Lightfoot	3400	1 KB	32-bit
JOP	2100	3 KB	32-bit

Table 3. JVM performance compared

Processor		Execution time	Relative performance
486SX25	Interpreting	19.55 s	1.00
486SX25	JVM with JIT	5.00 s	3.91
JOP		1.73 s	11.3

5.1 Applications

Balfour Beatty Austria has developed a *Kippfahrleitung* to speed up loading and unloading of goods wagons. The solution is to tilt the contact wire up on a line up to one kilometer. An asynchrony motor on each mast is used for this tilting. Nevertheless, it has to be done synchronic on the whole line. Technical this is a distributed embedded real-time control system with one processor board per mast communicating over an RS485 bus with a base station. The main challenge was to react to the sensors in real-time and control the bus access of this distributed system.

A second application of JOP is currently under development: The Austrian Railways adds a new security system for single-track lines. Every locomotive will be equipped with a GPS receiver and communication device. The position of the train, differential correction data for GPS and commands are exchanged with a server in the central station over a virtual private network. JOP is the heart of the communication device in the locomotive.

6 Conclusion

This paper presented the architecture of a hardware implementation of the JVM. The flexibility of FPGAs and HW/SW co-design makes it possible to adapt the resource usage of the processor for different applications. Predictable execution time of bytecodes, a predictable instruction cache and a simple extension of Java enable usage of JOP in real-time applications. Although the full implementation of the JVM is still missing, one successful project showed that JOP is mature enough to be used in real-world applications. JOP encourages usage of Java in embedded systems.

Additional work has to be done to complete the JVM and port essential parts of the Java library. Further research will focus on the predictable instruction cache and hardware support for the real-time extensions of Java. More information and all VHDL and Java sources for JOP can be found in [16].

References

1. K. Arnold and J. Gosling. *The Java Programming Language*, Addison Wesley, 2nd edition, 1997.
2. Bollela, Gosling, Brosgol, Dibble, Furr, Hardin and Trunbull. *The Real-Time Specification for Java*, Addison Wesley, 1st edition, 2000.
3. T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*, Addison Wesley, 2nd edition, 1999.
4. Altera Corporation. *ACEX Programmable Logic Family*, Data Sheet, ver. 1.01, April 2000.
5. Sun microsystems. *picoJava-II Processor Core*, Data Sheet, April 1999.
6. P. Puschner and C. Koza. Calculating the Maximum Execution Time of Real-Time Programs, *Real-Time Systems Journal*, 1(2): pp. 159–176, September 1989
7. G. Bernat, A. Burns and A. Wellings. Portable Worst-Case Execution Time Analysis Using Java Byte Code, In *Proc. 6th Euromicro conference on Real-Time Systems*, pp. 81–88, June 2000
8. Bate, G. Bernat, G. Murphy and P. Puschner. Low-Level Analysis of a Portable Java Byte Code WCET Analysis Framework, In *6th IEEE Real-Time Computing Systems and Applications (RTCSA2000)*, pp. 39–48, South Korea, December 2000
9. P. Puschner and A. J. Wellings. A Profile for High Integrity Real-Time Java Programs. In *Proc. of the 4th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, 2001
10. Burns and B. Dobbing. The Ravenscar Tasking Profile for High Integrity Real-Time Programs. In *Proc. of the 1998 annual ACM SIGAda international conference on Ada*, pp. 1–6, Washington, D.C., United States, 1998
11. J. Kwon, A. Wellings and S. King. Ravenscar-Java: a high integrity profile for real-time Java, In *Proc. of the 2002 joint ACM-ISCOPE conference on Java Grande*, pp. 131–140, Seattle, Washington, USA, 2002
12. Corsaro, D. Schmidt. The Design and Performance of the jRate Real-Time Java Implementation. Appeared at *the 4th International Symposium on Distributed Objects and Applications*, 2002
13. Altera Corporation. *Nios Soft Core Embedded Processor*, Data Sheet, ver. 1, June 2000.
14. M. Delvai, W. Huber, P. Puschner and A. Steininger. Processor Support for Temporal Predictability – The SPEAR Design Example. In *Proc. of Euromicro Conference on Real-Time Systems (ECRTS03)*, Porto, Portugal, July 2003.
15. Xilinx Corporation. *Lightfoot 32-bit Java Processor Core*, Data Sheet, September 2001.
16. Martin Schoeberl. JOP – a Java Optimized Processor, <http://www.jopdesign.com>.

Processing Group Parameters in the Real-Time Specification for Java

A. Burns and Andy J. Wellings

Department of Computer Science, University of York
YO10 5DD, U.K.
{burns,andy}@cs.york.ac.uk

Abstract. The Real-Time Specification for Java (RTSJ) does not support any particular aperiodic server technology. Instead it provides processing group parameters, which allow a group of schedulable objects to be allocated a capacity and a replenishment period. However, no guidelines are given on how to use them. Furthermore, there are currently no appropriate schedulability analysis techniques which can be used to model the full generality of the RTSJ model. This paper proposes restrictions on the use of processing group parameters and discusses the problems of producing the associated analysis techniques.

1 Introduction

In any system where it is required to give guarantees, aperiodic activities present a problem. As they have no well-defined release characteristics, they can impose an unbounded demand on the processor's time. If not handled properly, they can result in periodic or sporadic schedulable objects (real-time threads and asynchronous event handlers) missing their deadlines, even though those schedulable objects have been "guaranteed".

One simple way of scheduling aperiodic activities, within a pre-emptive priority-based scheme, is to run them at a priority below the priorities assigned to periodic and sporadic schedulable objects [3]. In effect, the aperiodic schedulable objects run as background activities and, therefore, cannot pre-empt the other schedulable objects. Although a safe scheme, this does not provide adequate support to aperiodic schedulable objects that will often miss their target completion times if they only run as background activities. To improve the situation, a server can be employed. Servers protect the processing resources needed by periodic and sporadic schedulable objects but otherwise allow aperiodic schedulable objects to run as soon as possible. Several types of servers have been defined by the real time community. The major ones for fixed priority based systems are: deferrable servers[5] and sporadic servers [7].

With the deferrable server, an analysis is undertaken (using, for example, the response time approach [6]) that enables a new activity to be introduced at the highest priority¹. This process, the server, thus has a period, $\bar{T}_s$ and a

¹ Servers at other priorities are possible but the description is more straightforward if the server is given a higher priority than all the hard processes.

capacity $\bar{C}_s$. These values are chosen so that all the hard processes in the system remain schedulable even if the server executes periodically with period $\bar{T}_s$ and execution time $\bar{C}_s$. At run-time, whenever an aperiodic process arrives, and there is capacity available, it starts executing immediately and continues until either it finishes or the capacity is exhausted. In the latter case, the aperiodic process is suspended (or transferred to a background priority). With the deferrable server model the capacity is replenished every $\bar{T}_s$ time units.

The operation of the sporadic server differs from the deferrable server in its replenishment policy. With the sporadic server, if an aperiodic activity arrives at time t and uses c capacity then the server has this c capacity replenished $\bar{T}_s$ time units after t . In general, sporadic servers can furnish higher capacity than deferrable servers but have increased implementational overheads. Both can be analysed using response time analyses [2]. Sporadic servers are supported by POSIX [4].

The Real-Time Specification for Java (RTSJ) does not support any particular aperiodic server technology. Instead it provides processing group parameters, which allow a group of schedulable objects to be allocated a capacity and a replenishment period. However, no guidelines are given on how to use them and there is no indication of the appropriate schedulability analysis techniques to be used. Indeed, to our knowledge, there are currently no appropriate schedulability analysis techniques which can be used to model the full generality of the RTSJ model.

In section 2 of this paper, an overview of scheduling in the RTSJ is presented along with the detailed semantics of processing group parameters. Section 3 then proposes two restrictive usages: the first supports a simple deferrable server for handling aperiodic activities, the second considers groups consisting of periodic and sporadic schedulable objects. Section 4, then develops the schedulability analysis for the latter case. Finally, section 5 presents conclusions.

2 Overview of Scheduling in the RTSJ

The RTSJ incorporates the notion of a schedulable object rather than considering just threads. A schedulable object is any object that implements the `Schedulable` interface. The current specification essentially provides two types of object that implement this interface, `RealtimeThreads` and `AsyncEventHandlers`. Objects that implement the `Schedulable` interface have the following associated attributes (represented by classes).

ReleaseParameters – Giving the processing cost for each release (its CPU budget) of the schedulable object and its deadline; if the object is released periodically or sporadically then subclasses allow an interval to be given. Event handlers can be specified for the situation where the deadline is missed or the processing resource consumed becomes greater than the cost specified. However, note that there is no requirement for a real-time JVM to monitor the processing time consumed by a schedulable object. If it does, then there is a requirement that a schedulable object be given no more than its cost in processing units each

release. It should be noted that Version 1 of the RTSJ makes no mention of blocking time in any of the parameters associated with schedulable objects. The assumption is that a particular implementation will subclass `ReleaseParameters` to bring in this data. Typically, this will be a relative time value set by the programmer as a result of off-line analysis of the code. It is likely that Version 1.1 will formally introduce blocking time into the `ReleaseParameters` class.

SchedulingParameters – The `SchedulingParameters` class is empty; however subclasses allow the priority of the object to be specified along with its importance to the overall functioning of the application. Although the RTSJ specifies a minimum range of real-time priorities (28), it makes no statement on the allowed values of the importance parameter.

MemoryParameters – Giving the maximum amount of memory used by the object in its default memory area, the maximum amount of memory used in immortal memory, and a maximum allocation rate of heap memory. An implementation of the RTSJ is obligated to enforce these maximums and throw exceptions if they are violated.

ProcessingGroupParameters – This allows several schedulable objects to be treated as a group and to have an associated period, cost and deadline (see Section 2.1).

The methods in the `Schedulable` interface can be divided into three groups.

- Methods that will communicate with the scheduler and will result in the scheduler either adding or removing the schedulable object from the list of objects it manages (called its feasibility set), or changing the parameters associated with the schedulable object (but only if the resulting system is feasible).
- Methods that get or set the parameter classes associated with the schedulable object. If the parameter object set is different from the one currently associated with the schedulable object, the previous value is lost and the new one will be used in any future feasibility analysis performed by the scheduler. Note, these methods do not result in feasibility analysis being performed and the parameters are changed even if the resulting system is not feasible.
- Methods that get or set the scheduler. For systems that support more than one scheduler, these methods allow the scheduler associated with the schedulable object to be manipulated.

The only scheduler that the RTSJ fully defines is a priority scheduler, which can be summarized as having:

Scheduling policy. The priority scheduler

- supports the notion of base and active priority;
- orders the execution of schedulable objects on a single processor according to the active priority;
- supports a real-time priority range of at least 28 unique priorities (the larger the value, the higher the priority);
- requires the programmer to assign the base priorities (say, according to the relative deadline of the schedulable object);

- allows base priorities to be changed by the programmer at run time;
- supports priority inheritance or priority ceiling emulation inheritance for synchronized objects;
- assigns the active priority of a schedulable object to be the higher of its base priority and any priority it has inherited.

Scheduling mechanism. The priority scheduler

- supports pre-emptive priority-based dispatching of schedulable objects - the processor resource is always given to the highest priority runnable schedulable object;
- does not define where in the run queue (associated with the priority level) a pre-empted object is placed; however, a particular implementation is required to document its approach and the RTSJ recommends that it be placed at the front of the queue;
- places a blocked schedulable object that becomes runnable, or has its base priority changed, at the back of the run queue associated with its (new) active priority;
- places a schedulable object which performs a `Thread.yield()` method call at the back of the run queue associated with its priority
- does not define whether schedulable objects of the same priority are scheduled in FIFO, round-robin order, or any other order.

Schedulability (feasibility) analysis. The `PriorityScheduler` requires no particular analysis to be supported.

2.1 Processing Group Parameters

The RTSJ provides support for aperiodic server technologies via processing group parameters. When processing group parameters are assigned to one or more schedulable objects, a logical server is created. The server's start time, cost (capacity) and period are defined by the particular instance of the parameters. These collectively define the points in time when the server's capacity is replenished.

Any aperiodic schedulable object that belongs to a processing group is executed at its defined priority. However, it only executes if the server still has capacity (and the object has not overrun its own individual CPU budget). As it executes, each unit of CPU time consumed is subtracted from the server's capacity (as well as its own). When capacity is exhausted, the aperiodic schedulable objects are not allowed to execute until the start of the next replenishment period.

The RTSJ facilities are very general. They allow

- schedulable objects of different priorities to be assigned to the same group,
- the inclusion of sporadic and periodic schedulable objects,
- the schedulable objects to also have individual costs and deadlines,
- the “servers” to be given a deadline, and cost overrun and deadline miss handlers.

The latter may be useful if the programmer has more information about the pattern of arrivals of the aperiodic activities and wants to detect variations from that pattern.

The full definition of the `ProcessingGroupParameters` class is given below.

```
package javax.realtime;
public class ProcessingGroupParameters {
    // constructors
    public ProcessingGroupParameters( HighResolutionTime start,
        RelativeTime period, RelativeTime cost,
        RelativeTime deadline,
        AsyncEventHandler overrunHandler,
        AsyncEventHandler missHandler);

    // methods
    public RelativeTime getCost();
    public AsyncEventHandler getCostOverrunHandler();
    public RelativeTime getDeadline();
    public AsyncEventHandler getDeadlineMissHandler();
    public RelativeTime getPeriod();
    public HighResolutionTime getStart();
    public void setCost(RelativeTime cost);
    public void setCostOverrunHandler(AsyncEventHandler handler);
    public void setDeadline(RelativeTime deadline);
    public void setDeadlineMissHandler(AsyncEventHandler handler);
    public void setPeriod(RelativeTime period);
    public void setStart(HighResolutionTime start);
}
```

3 Using Processing Group Parameters

Processing group parameters allow very general servers to be set up. However, to get full predictability, it will be necessary for implementations to restrict the generality so that they support one (or more) of the well-known server technologies, such as deferrable servers. This section considers the various ways in which processing group parameters can be used.

3.1 Handling Aperiodics

The simplest form of restriction on processing group parameters is to provide a subclass of `ProcessingGroupParameters` so that the scheduler can support, say, deferrable servers.

```
import javax.realtime.*;
public class DeferrableProcessingGroupParameters extends
    ProcessingGroupParameters {

    public DeferrableProcessingGroupParameters(
```

```

        HighResolutionTime start, RelativeTime period,
        RelativeTime cost, PriorityParameters pri) {
    super(start, period, cost, period, null, null);
    // save priority etc, throw exceptions
}
...
}

```

The scheduler would then throw exceptions if schedulable objects having the same deferrable processing group parameters do not have the same priority and do not have aperiodic release parameters. Standard schedulability analysis can be used to model the impact that a particular group will have on the rest of the system.

Note also, that cost overruns and deadline misses of the individual aperiodic activities will also be detected.

The advantages of this approach include: it models one of the standard server technologies, it is simple to implement, and it uses standard analysis techniques.

The disadvantage is that aperiodic activities are handled, effectively, in FIFO order.

A simple extension of the above allows the aperiodic activities to have different priorities. However, given that there are no bounds on the release of each of the aperiodic schedulable objects, the analysis must assume that the highest priority one is always released. Hence, the server is considered to run at the highest priority.

3.2 Handling Periodic and Sporadics

Whilst it might seem curious to want to handle periodic activities with servers, it is useful for the situation where there is a subsystem containing some periodic schedulable objects which have a large variation between their average and worst case execution time. In this situation, assuming that each object performs to its worst case at the same time as every other object of that system is very pessimistic. Using processing group parameters it is possible to

- bound the interference of the group to other schedulable objects in the system, thereby ensuring that the latter will not be impacted if the worst case phasing does occur
- allow each schedulable object to obtain its maximum CPU time when it needs it.

Including sporadic schedulable objects into a processing group ensures that they do not occur more frequently than estimated.

Of course, with this approach it may not be possible to give guarantees to all the schedulable objects in the group. However, it is possible to calculate their individual response times and the impact that the group has on other schedulable objects in the system. See section 4.

The following class illustrated how subsystems might be represented.

```

import javax.realtime.*;
public class SubsystemProcessingGroupParameters extends
    ProcessingGroupParameters
{
    public SubsystemProcessingGroupParameters(
        HighResolutionTime start,
        RelativeTime period, RelativeTime cost)
    {
        super(start, period, cost, period, null, null);
    }
    ...
}

```

4 Response-Time Analysis for Subsystem Group Parameters

The analysis in this section assumes that the application under consideration consists of a set of periodic or sporadic schedulable objects ², e.g. τ_i . Each τ_i has a minimum inter-arrival time of T_i , maximum computation time, C_i , and deadline D_i ($D_i \leq T_i$). We need to verify that the worst-case response-time, R_i is always less than its deadline ($R_i \leq D_i$). We assume that each τ_i has a unique priority P_i and is a member of exactly one server group ³. Each server S_s has a capacity $\bar{C}_s$ and a replenishment period $\bar{T}_s$. For simplicity we assume each server's deadline is equal to $\bar{T}_s$. We also assume that τ_i does not share data with any object in another server (although it may do so with schedulable objects within its own server).

If, as will be the case in general, τ_i executes over a number of server periods it is not possible to calculate R_i exactly (it is a NP-hard problem). This is due to (at least) two properties of the computation model:

- The worst-case does not necessary occur when all higher priority objects are released at the same time (the critical instance).
- The worst-case does not necessarily occur when sporadic objects arrive at their maximum frequency.

Both of these properties make the problem of calculating an exact R_i intractable. In the following analysis therefore an approximation is produced that gives an upper-bound for R_i . This analysis allows the feasibility of an application to be tested and, perhaps more importantly, allows the designer to explore different values for the server parameters ($\bar{C}_s$, $\bar{T}_s$).

² It is trivial to include aperiodic schedulable objects represented by deferrable servers into this model.

³ A schedulable object which is not a member of a group is modeled as the sole member of a group whose release parameters are identical to the schedulable object's release parameters.

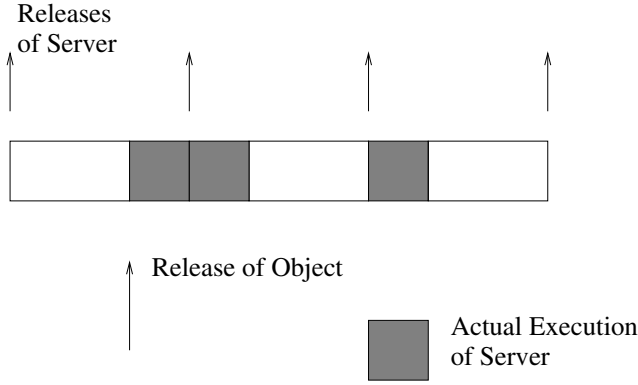


Fig. 1. Worst-Case Release of External Server

Although the parameter $\bar{C}_s$ is described as the server's capacity it is not guaranteed. It may be the case that S_s is never allocated that amount of execution time in any $\bar{T}_s$. Note that any capacity not used in $\bar{T}_s$ is lost. What $\bar{C}_s$ represents is the maximum capacity that the server is allowed to assign to the objects it serves. To calculate R_i , we need to compute the guaranteed capacity each server ($\tau_i \in S_s$) will obtain in any $\bar{T}_s$. As the server executes at the priority of its clients we need to obtain $\hat{C}_s(i)$, the guaranteed capacity of S_s at priority level P_i (note $\hat{C}_s(i) < \bar{C}_s$ - it is capped by the server's capacity).

To evaluate $\hat{C}_s(i)$ we need to calculate the maximum interference from other servers and other objects within S_s . First other servers: analysis of deferred servers [2] indicate that the worst-case occurs when $\hat{C}_s(i)$ is available for interference at the very end of a server's period and again at the start of the following period. This is illustrated in Figure 1.

If the server (S_ρ) was always interfering with τ_i then in any interval $[0 \ t)$ the interference would be

$$\left\lceil \frac{t + (\bar{T}_\rho - \bar{C}_\rho)}{\bar{T}_\rho} \right\rceil \bar{C}_\rho. \quad (1)$$

But this is an upper-bound. Server S_ρ will only interfere if it has objects with higher priority than P_i .

The usual response time equation has the following term for interference:

$$\sum_{\tau_m \in hp(i, \rho)} \left\lceil \frac{t}{T_m} \right\rceil C_m, \quad (2)$$

where $hp(i, \rho)$ is the set of schedulable objects with priority greater than P_i in server S_ρ .

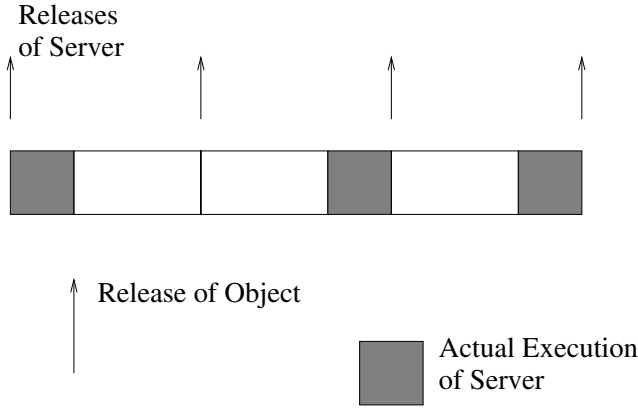


Fig. 2. Worst-Case Release of Own Server

Combining (1) and (2) we get a formulae for the interference from server S_ρ ($\tau_i \notin S_\rho$):

$$I_s(i, t) = \min \left(\left\lceil \frac{t + (\bar{T}_\rho - \bar{C}_\rho)}{\bar{T}_\rho} \right\rceil \bar{C}_\rho, \sum_{\tau_m \in hp(i, \rho)} \left\lceil \frac{t}{T_m} \right\rceil C_m \right). \quad (3)$$

The usual response time equation is solved for a particular computation time C . In this formulation we wish to obtain the maximum c that will be guaranteed in $\bar{T}_s$. That is

$$r = c + \sum I_s(i, r), \quad (4)$$

where the sum is over all the servers apart from S_s ($\tau_i \in S_s$).

This is solved to give $r \leq \bar{T}_s$ - but $(c + 1)$ would produce a value of r greater than $\bar{T}_s$. There are various branch and bound methods of iterating towards this value of c [1]. Once c is calculated then this is the required value of $\hat{C}_s(i)$.

To return to the problem of calculating the response time R_i of object τ_i we need to allocate $\hat{C}_s(i)$ to the requirements of τ_i , namely C_i . Unfortunately in the worst-case the full capacity of the server S_s has been used up just before τ_i arrives, and further $\bar{C}_s(i)$ amounts are only available at the end of the server's replenishment period. This is illustrated in Figure 2.

The final issue to note is that data sharing via some appropriate priority inheritance protocol will mean that $C_i + B_i$ (where B_i is the blocking term in the standard analysis) needs to be guaranteed. So let V be defined by

$$V = \left\lceil \frac{C_i + B_i}{\hat{C}_s(i)} \right\rceil \quad (5)$$

then

$$R_i \leq (\bar{T}_s - \bar{C}_s) + V\bar{T}_s. \quad (6)$$

This is a safe upper bound. Some improvements can be obtained by:

- Being more exact about the maximum capacity that could have been removed prior to being released (as depicted in Figure 2) - for low priority objects there may not exist even lower priority objects to ‘steal’ the capacity.
- Also, as illustrated in Figure 2, the guaranteed capacity may not be right at the end of the period – the value of r in equation (4) would give a tighter completion time.
- The amount of guarantee required in the final period is likely to be less than a full $\hat{C}_s(i)$ - again solving equation (4) for the actual c required in this last period would give a tighter completion time.

All of these improvements will reduce the bound on R_i , but to reiterate the point made at the beginning of this section, a (general) tight bound on R_i is not possible to obtain. An analysis of specific application requirements is needed to judge the effectiveness of the analysis represented here.

5 Conclusions and Future Work

The RTSJ has introduced a novel approach to capacity sharing between schedulable objects. In its full generality, the model is too complex to develop supporting analysis techniques. This paper has proposed two restrictive usages. One for handling groups of aperiodic activities and the other for handling groups of periodic and sporadic activities. The former can be supported by standard deferrable server analysis techniques. The latter needs new analysis techniques. This paper has discussed the problems of developing these techniques and goes some way to solving them. Future work will include: verifying the approach by simulation and extending the model to allow for communication between schedulable objects in different groups.

Acknowledgement. The authors gratefully acknowledge the contribution of Guillem Bernat to some of the ideas expressed in this paper.

References

1. G. Bernat. Response time analysis of asynchronous systems. *Real-Time Systems*, to appear.
2. G. Bernat and A. Burns. New results on fixed priority aperiodic servers. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 68–78, 1999.
3. A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 3rd edition, 2001.
4. IEEE. Portable operating system interface: Amendment d: Additional real-time extension [C language]. IEEE/1003.1d, IEEE, 1999.
5. J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in a hard real-time environment. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 261–270, 1987.

6. J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.
7. B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings IEEE Real-Time Systems Symposium*, pages 251–258, 1988.

Doing Firm-Real-Time with J2SE APIs

Kelvin Nilsen

CTO, Aonix NA, 877 S. Alvernon Way, Suite 100, Tucson, AZ 85711
kelvin@aonix.com

Abstract. Today's real-time developers face a variety of challenges covering a broad spectrum of requirements. No single Java "standard" best suits the needs of all developers. Using appropriate implementation techniques, "standard edition" Java makes an excellent platform for development of large and complex firm-real-time systems. Compared to so-called "real-time extensions" for Java, this approach offers much easier development and maintenance, greater portability, scalability and flexibility, and improved dynamic behavior.

1 Introduction

The Java programming language offers the potential of greatly reducing the effort required to develop and maintain embedded real-time software. Developers of traditional desktop and enterprise applications often realize two-fold productivity improvements when switching to Java from C++. And developers responsible for integration of independently developed software components have experienced up to a ten-fold productivity boost when working with Java vs. C++ components [1]. By using standard-edition Java APIs, many of these same benefits can be realized by developers of firm-real-time systems as well.

We use the term *firm real-time* to emphasize that we are not speaking of *soft real-time* as occasionally understood to mean: "Qué será será; whatever will be, will be". Rather, we are speaking of disciplined development in which software engineers carefully analyze deadlines, resource requirements, and schedulability. We differentiate from hard-real-time in that, for the most part, resource requirements are determined empirically, by measuring the behavior of individual components rather than theoretical analysis. Generally, empirical evaluation provides statistical confidence but does not offer absolute guarantees. We reserve the term *hard-real-time* to describe systems which are proven through mathematical analysis to always meet all deadlines. Such proofs, which necessarily depend on intimate knowledge of the RTOS implementation, the CPU's pipeline and cache architecture, the organization of the memory subsystem, and the compiler's code generation model, are extremely difficult and are generally only practical for very small and simple real-time systems. We are not aware of any hard-real-time Java implementations today, though we are involved in an effort to create one such system to support safety-critical development [2, 3].

Since it was first publicly released in 1996, Java has evolved into several different versions to serve the needs of different audiences. J2SE is the "standard edition" that is used by nearly all Java developers [4]. This is the version that typically runs on

personal desktop computers, and this is the version that almost all of the software components available for licensing from 3rd parties or downloadable as open-source components. J2EE is the enterprise edition [5]. It includes all of the J2SE libraries, and has additional server-specific libraries to support, for example, secure transaction processing. J2ME is the microedition, which is available in two configurations, the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC) [6]. J2ME is designed for memory-limited systems like cell phones and set-top boxes. Developers targeting one of these J2ME standards have to carefully avoid dependencies on standard Java libraries such as RMI (remote method interface), and JDBC (data base connectivity).

RTSJ is the Real-Time Specification for Java [7]. This is a set of extensions that can be combined with any of the existing Java platforms (J2ME, J2SE, or J2EE) to allow development of real-time software in “Java”. To date, RTSJ implementations run only with J2ME and only with the Linux operating system. Further, today’s developer of RTSJ-style no-heap real-time threads is restricted to a subset of the J2ME libraries, cannot use automatic garbage collection, and must adhere to restrictive memory-usage guidelines in order to obtain real-time threading behavior. Though it is possible for an RTSJ implementation to provide real-time garbage collection, the RTSJ specification does not require this and in fact several attributes of the RTSJ specification make it more difficult to implement incremental paced garbage collection. Based on technologies available today, the RTSJ developer cannot invoke off-the-shelf Java software components from their real-time threads and real-time RTSJ components are generally not portable across operating systems or between different compliant RTSJ implementations.

In contrast, this paper focuses on an alternative approach to real-time development with the Java programming language. It describes use of the PERC real-time Java virtual machine [8]. PERC is a clean-room implementation of headless J2SE, supporting all of J2SE except graphics (AWT and Swing). It is targeted to the needs of developers who are attracted to the high-level benefits of Java and have firm-real-time requirements ranging from one to tens of milliseconds. It has been successfully deployed in a variety of applications in domains such as network infrastructure equipment, manufacturing automation, energy production, and commercial transportation.

The PERC run-time environment, like Java itself, represents a *de facto* standard. The semantics of the PERC APIs, garbage collector, and task scheduler are not currently described in any formal standards document. However, they are well documented in company literature and many developers have successfully ported real-time applications across a broad assortment of operating systems and CPU platforms by building their software upon the *de facto* PERC standard. It should be noted that there are no intellectual property barriers to prevent other companies from implementing the same semantic standards in their J2SE-compatible products or from joining with Aonix in taking this *de facto* standard to an international standardization body.

Similar to the RTSJ specification, which allows each implementation of the RTSJ to independently define the basis for real-time development (number of priorities, synchronization semantics, wait queue ordering, library compatibility, workload admission testing algorithms, real-time scheduling strategies, and I/O interruption semantics), the J2SE specification also allows individual implementors to constrain the behavior of the run-time environment to empower real-time development. The

PERC run-time controls the scheduling of Java threads to ensure consistent fixed-priority dispatching (without priority aging) and priority inheritance across all platforms. PERC offers paced real-time garbage collection with high reliability achieved through accurate scanning and automatic defragmentation of the memory heap. Since PERC behaves the same on all of its supported platforms,¹ it delivers on Java's original WORA promise. This enables cross-platform development and testing (develop on a workstation and deploy on an embedded board), straightforward integration of 3rd party Java software components into real-time applications, and composability of independently developed real-time Java components. To date, PERC has been ported to ETS, INTEGRITY, Linux, LynxOS, Nucleus, OSE, OSE Softkernel, RTX, VxWorks, VxWorks AE, VxSim, WinCE and Windows NT operating systems. It runs on ARM, MIPS, Pentium, PowerPC, XScale, and 68K. The PERC JIT (just-in-time) and AOT (ahead-of-time) compilers are available for ARM, Pentium, PowerPC, and XScale.

2 Implementation Choices and Semantic Guarantees

The internals of the PERC virtual machine were designed to support deployment of highly reliable predictable real-time systems. This section describes some of the special implementation choices and the resulting semantic guarantees that PERC offers to the real-time developer. Note that, unlike the RTSJ, PERC supports the same real-time-enhanced semantics for standard J2SE libraries across all platforms. With the RTSJ, programs are portable between compliant RTSJ implementations only if both implementations choose to offer the same libraries and the same semantics for thread priorities and synchronization. With both PERC and RTSJ, portability depends in part on availability of resources.

2.1 Garbage Collection

Reliability. All commercially available Java virtual machines collect garbage, but some implementations do it better than others. Many garbage collectors make conservative approximations of which memory is still in use. This means they cannot guarantee to reclaim all of the dead memory in the system and usually cannot defragment memory by relocating in-use objects to consecutive memory locations. Note that a single dead object that is improperly identified as live may hold references to an arbitrarily large collection of additional dead objects, all of which must be conservatively treated as live. Since a conservative or partially conservative garbage collector cannot guarantee to reclaim dead memory, a Java application running on such a system cannot assure the absence of memory leaks.

¹ PERC's ability to "behave the same" on all operating systems is restricted to Java threading behavior. To the extent that developers incorporate native code, the native code introduces an aspect of non-portability. To the extent that the operating system's device drivers or higher-priority threads steal CPU time away from the PERC virtual machine, PERC's ability to meet real-time constraints may be compromised.

Another important aspect of garbage collection reliability is defragmentation of the memory heap. Since the garbage collector knows which objects are live and it knows exactly which other objects make direct reference to this object, it can relocate live objects to contiguous memory locations, updating all the pointers that refer to each of the relocated live objects. By so doing, the garbage collector is able to coalesce what was once an assortment of scattered free segments into a single large free segment. Without these sorts of defragmentation capabilities, Java VM reliability suffers because even though a considerable amount of free memory might exist, the VM may not be able to find any free segment large enough to satisfy a particular allocation request.

Real-Time Garbage Collection. The act of garbage collection must traverse the graph of Java objects that are reachable by following some chain of pointers originating with a VM-specific “root pointer”. There are several key requirements that must be satisfied by any real-time garbage collector:

1. It must be possible to quickly preempt the garbage collection task. Note that the time required to complete a garbage collection pass is proportional to the size of the heap. Typical configurations of currently available commercial Java virtual machines require tens of seconds of CPU time to perform a complete garbage collection pass.
2. For real-time systems that must allocate memory on the fly (under real-time constraints), it is essential that garbage collection progress be paced against the application’s ongoing need for memory allocation. This requires that the total CPU time required to complete garbage collection be bounded and that garbage collection algorithms be incremental so that following each preemption, garbage collection resumes where it left off rather than restarting from scratch.

Mostly stationary garbage collection. One of the innovations introduced in the PERC virtual machine is its patent-protected real-time garbage collection system [9]. The PERC garbage collector divides its effort into thousands of small uninterruptible increments of work. Depending on the choice of underlying CPU, the maximum time required to execute an increment of garbage collection is approximately 100 microseconds. When garbage collection resumes following preemption by a higher priority application thread, it resumes where it left off. There is never a need to go back and restart any phase of garbage collection.

Figure 1 illustrates the incremental copying garbage collection technique used within the PERC virtual machine. At the start of garbage collection, *from-space* contains the three live objects **A**, **B**, and **C** and *to-space* is empty. Garbage collection consists of incrementally reserving space and subsequently relocating each of the live objects. Any attempt to access the object during garbage collection is automatically redirected to the single valid copy of the object. At the time this snapshot was drawn, the valid versions of objects **B** and **C** are **B'** and **C'** respectively. The valid version of **A** is **A** itself, because this object has not yet been relocated. Each of the invalid versions maintains a pointer to the valid version, represented by the arrows from **A'** to **A**, **B** to **B'** and **C** to **C'**. Objects waiting to be relocated, such as **A**, maintain a

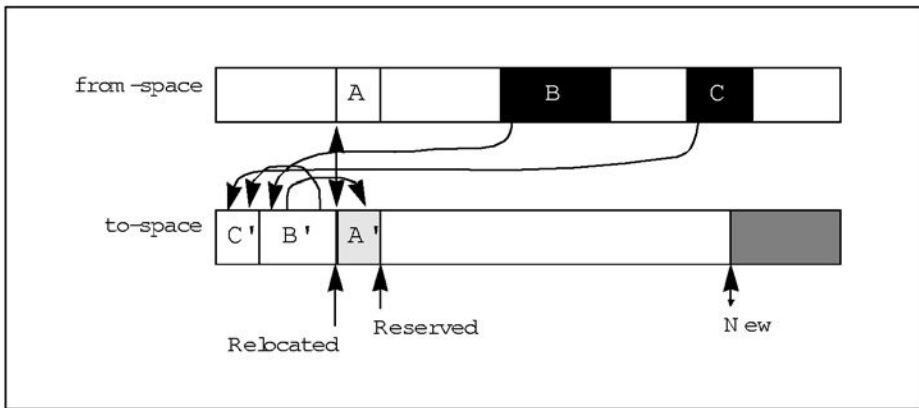


Fig. 1. Incremental Copying Garbage Collection

forwarding pointer to the memory that has been reserved to hold the eventual copy (e.g. **A'**). This allows other references to **A** to be eventually replaced with references to **A'**. As objects are being relocated, each pointer contained within the object is replaced by a pointer to the new *to-space* copy of the referenced object. Thus, object **B'** holds pointers to **A'** and **C'**, whereas object **B** held pointers to **A** and **C**. A beneficial side effect of copying garbage collection is that the unused memory scattered throughout *from-space* is coalesced into a single larger free segment from which new memory requests that are issued while garbage collection is taking place can be served.

Incremental mark-and-sweep garbage collection is a two-phase technique. The objective of the first phase is to mark each reachable object by linking it onto the *scan list*. During this phase, the garbage collector repeatedly removes the leading object from the scan list (by advancing the scan-list head pointer) and scans each of its pointer fields in order to mark the objects it references. Even though we advance the scan-list head pointer after the lead object on the queue has been scanned, we leave its scan-link field unchanged. We do this to remember that this object has already been marked. Once an object is marked, we do not put it back on the scan list even if we discover additional references to it. Note that scanning of individual objects is itself incremental. This phase ends when there are no more objects on the scan list.

The second phase consists of sweeping through memory from low to high address. For each address we examine, we know that it is either the start of a marked object, the start of an unmarked object, or the start of a free segment. Unique bit patterns in the object headers allow us to differentiate between the three possibilities. Header information also identifies the size of each object, enabling the sweep process to skip over the internals of each examined memory segment. Sweep treats each of the three situations differently. If it is looking at a marked object, it simply clears the mark field in preparation for the next garbage collection pass. If it is looking at a free segment, it coalesces this free segment with the preceding object if that also is a free segment. If it is looking at an unmarked object, it converts this into a free segment and then coalesces it with the preceding object if that is a free segment. Though we do

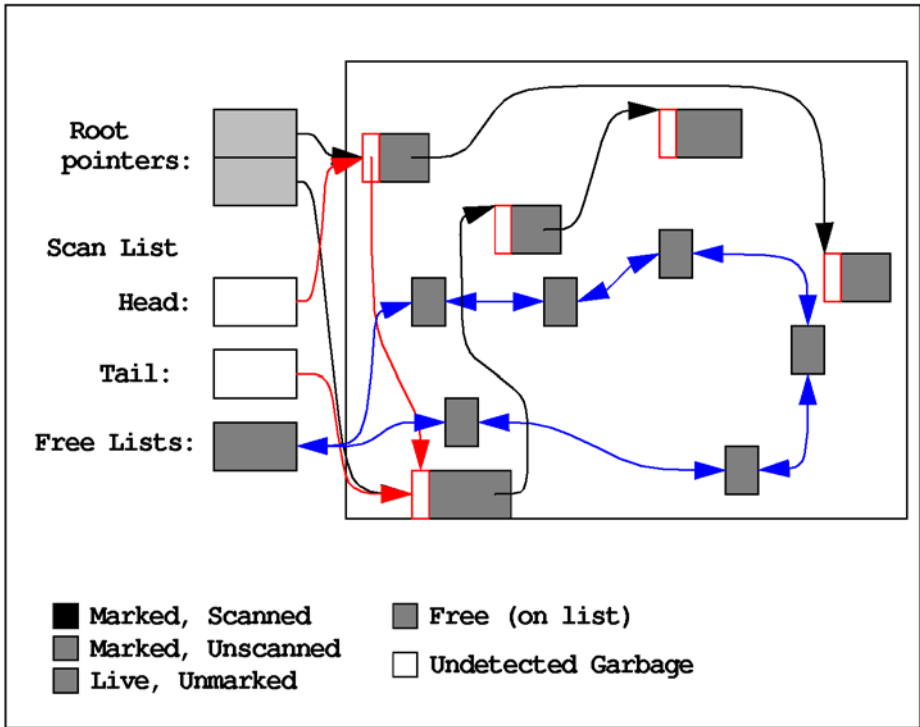


Fig. 2. Incremental Mark-and-Sweep Garbage Collection

not show it in the illustration, the PERC virtual machine actually maintains multiple free lists, each representing segments corresponding to a different class of sizes. Note that the free lists are doubly linked. This allows us to remove a segment from its corresponding list, coalesce it with a neighboring free segment, and insert the resulting larger free segment into its corresponding list in small constant time.

During incremental mark-and-sweep garbage collection, application threads that preempt the garbage collector may rearrange the relationship between objects before relinquishing to the garbage collector. This could potentially confuse the interrupted mark phase of garbage collection. To remedy this problem, application threads execute a *write barrier*. If garbage collection was marking when it was preempted, the application thread will automatically mark the referenced object each time it overwrites a pointer field.

The PERC virtual machine uses a hybrid technique which combines the benefits of incremental copying and mark-and-sweep garbage collection. It divides the memory allocation pool into multiple equal-sized regions and, at the beginning of each garbage collection pass, it selects two regions to serve as *to-* and *from-space* respectively. These regions are defragmented using the incremental copying garbage collection technique described above. The unused memory in the other regions is reclaimed using an incremental mark-and-sweep technique which does not relocate objects.

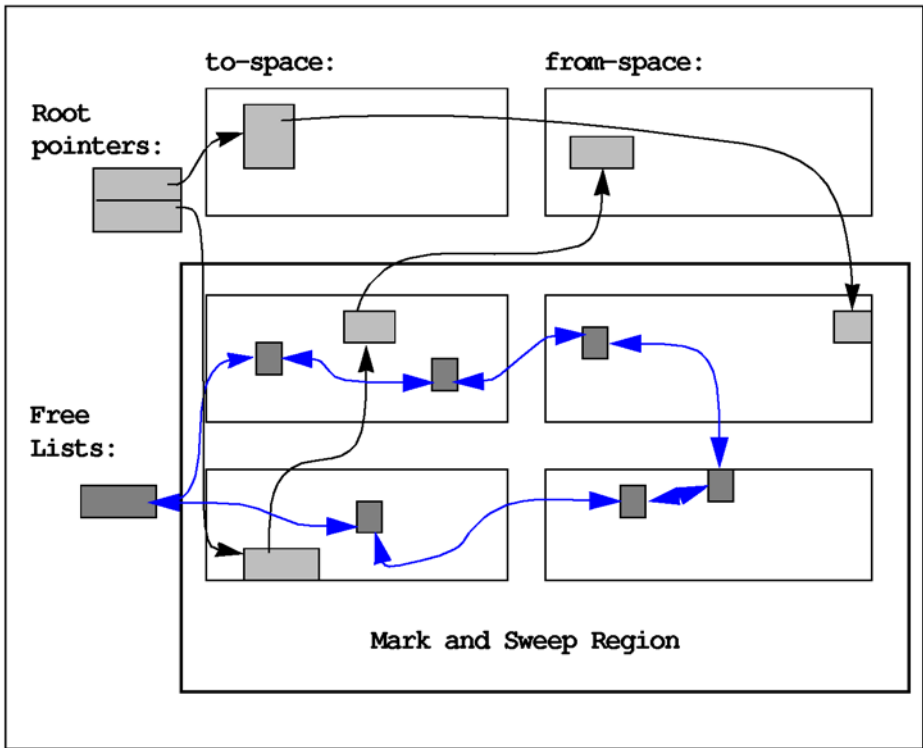


Fig. 3. Mostly Stationary Real-Time Garbage Collection

In typical situations, the mark-and-sweep technique achieves the highest memory utilization, but runs the risk of arbitrarily poor utilization in the rare event that it experiences severe memory fragmentation. Incremental copying garbage collection achieves guaranteed utilization of approximately 50%. Depending on workload characteristics and risk-reward profiles, users of the PERC virtual machine can configure the memory allocation pool for a small number of very large regions (with guaranteed defragmentation, but lower expected memory utilization) or a large number of relatively smaller regions.

Pacing of garbage collection. An important unique attribute of the PERC garbage collector is that the total effort required to complete garbage collection is bounded by a configuration-dependent constant, regardless of how much memory has recently been allocated or discarded, and independent of how many times the garbage collector is preempted by application threads. Given this property, it is straightforward to schedule garbage collection to periodically reclaim all of the dead memory in the system. The VM Management API allows garbage collection scheduling parameters to be adjusted on the fly in order to accommodate changes in the system workload. We call this *garbage collection pacing*. This makes sure that the system never exhausts its allocation free pool. If a virtual machine does not support pacing of garbage collection, then it is possible to experience situations in which a low-priority

task allocates memory subsequently desired by a high priority task, thereby forcing the high priority task to wait for garbage collection to complete before it can advance. This is an example of priority inversion that is nearly impossible to avoid with most implementations of the Java virtual machine, but is easily avoided with the PERC virtual machine.

2.2 Threading Behavior and Priority Inheritance

Regardless of operating system differences, PERC threads behave the same on all platforms. This is because the PERC virtual machine implements the synchronization locks (rather than relying upon the operating system to do so) and the PERC virtual machine takes full control over which PERC thread runs at any given instant in time. Furthermore, the PERC implementation of synchronization supports priority inheritance. This behavior is especially important when developing or testing on platforms that do not strictly honor programmer-specified priorities. Windows and Linux, for example, both age thread priorities, creating situations where high-priority threads occasionally yield to lower priority threads. This heuristic runs contrary to the desires and intentions of real-time programmers.

2.3 Improved Timer Services

Given that Java, as designed and distributed by Sun Microsystems, was not intended to support real-time programming, there are several timing-related APIs that the PERC JVM supplements in order to provide programmers with more precise control over time-constrained execution of Java software components.

1. The PERC VM supplies a **com.newmonics.util.Timer** class which implements all the same services as **java.util.Timer**, but provides slightly different semantics. Unlike **java.util.Timer**, the notion of time is maintained internal to the PERC virtual machine and is not affected if the operating system's real-time clock drifts or is modified by human operators.
2. The PERC VM associates a **com.newmonics.pvm.PercThread** object with each instance of **java.lang.Thread**. This provides access to additional time-related information for threads, such as the amount of CPU time spent in the thread. Use this to calculate the thread's execution time. **PercThread** also provides **sleepUntil()** and **waitUntil()** methods which can be used to implement non-drifting periodic execution and absolute timeouts.
3. PERC's enhanced timing services are measured in nanoseconds and take full advantage of the capabilities of the underlying hardware and operating system. Some platforms provide better timing resolution than others.
4. The PERC VM allows developers to set the tick period and the duration of each time slice.

2.4 The VM Management API

A Java virtual machine is a sophisticated run-time environment providing many more high-level services than typical minimalist real-time operating systems. Developers of embedded real-time Java applications occasionally need to be able to see and control what is happening beneath the hood. Achieving optimal system performance depends on finding appropriate balances between the memory and CPU-time budgets assigned to application threads and to certain background maintenance activities. By providing APIs to access and control this information, PERC makes it possible for software agents to take responsibility for self configuration of the embedded system. The garbage collection pacing agent to be bundled with PERC 4.1 is an example of one such agent. This component monitors trends in allocation rates, trends in live-memory retention, and trends in object longevity. It uses this statistical information to govern the rate at which garbage collection is performed. Its objectives are to dedicate to garbage collection exactly the amount of CPU time required to keep up with the application's appetite for new memory allocation, and no more, without ever consuming CPU time that had been previously set aside for use by time-constrained application threads. In overload situations, the pacing agent raises alert signals rather than interfering with higher priority real-time threads.²

Some examples of the services available by way of the VM Management API are listed below. Being able to examine and modify these system parameters is often essential in supporting cost-effective reliable real-time performance:

- Query and modify the maximum number of heap allocation regions.
- Query and modify the frequency and priority at which increments of garbage collection work are performed.
- Determine how much CPU time has been dedicated to execution of a particular Java thread.
- Determine which synchronization monitors are locked by particular threads, and which threads are waiting for access to particular synchronization monitors (to enable analysis of deadlock and resource contention bottlenecks).
- Query the RTOS priority at which a particular PERC thread is running.
- Query the duration of a PERC thread's tick period and time-slice duration. (Override the default values when you start up the PERC virtual machine.)
- Determine how much time the PERC virtual machine has been idle, and how much CPU time has been consumed at each priority level (to assist with rate-monotonic scheduling analysis).

² Note that the pacing agent is designed to automate and simplify configuration of the virtual machine based on certain assumptions regarding "reasonable" behavior of individual components. Aonix provides source code for the pacing agent so that developers can fine-tune its operation if they have different perspectives on what constitutes reasonable behavior. In situations where certain components are not trusted to behave reasonably, we recommend stronger abstractions, such as the time and memory partitioning techniques discussed briefly in numbered paragraph of Section.

3 Representative Applications

Many commercial applications have already been deployed and many others are under development based on the PERC platform. This section highlights some of the salient features that make firm-real-time J2SE appropriate for these applications.

1. Network infrastructure includes optical platforms, multi-service access devices, service concentration equipment, and IP infrastructure (gateways, routers, firewalls). Each of these “network elements” contains large amounts of software. One deployed optical networking product contains approximately 1 million lines of PERC code. This application domain needs high-level software abstraction to assure high programmer productivity and minimize introduction of software bugs, rapid evolution of software features, frequent dynamic reconfiguration of systems, and soft-real-time constraints measured in the tens of milliseconds.
2. Industrial automation uses computers to run manufacturing machinery and test equipment. PERC is most useful when applied to flexible manufacturing, distributed real-time control, and remote operation of systems. Typical real-time constraints for this domain range from 1 to tens of milliseconds.
3. Commercial telematics places computers in vehicles to assist with navigation, generate regulatory reports, track vehicle and driver performance, and facilitate communication between drivers and central dispatch. Remote installation of software components is especially important. Timing constraints are generally measured in hundreds of milliseconds.

4 Limitations and Plans for Future Work

Having reviewed the requirements for a large variety of proposed real-time Java applications during the seven and a half years that we have been supplying real-time Java technologies to the commercial marketplace, it is clear to this author that no single real-time Java “standard” will effectively serve all audiences. PERC is a good match for applications that have firm-real-time requirements ranging from 1 to 100 ms, typically comprised of tens to hundreds of thousands of lines of Java code, running on higher-end embedded processors with tens to hundreds of megabytes of RAM.

We do not recommend PERC for hard-real-time applications. Though it is theoretically possible to derive upper time bounds on all operations based on detailed knowledge of the application’s memory allocation behaviors and the underlying RTOS and CPU platform, we have never done this and we do not consider it economically feasible. The reasons developers are attracted to PERC for development of their firm-real-time application software are entirely inconsistent with traditional hard-real-time mindsets that demand, for example, full accountings from application developers for every byte of memory they will ever allocate, abstinence from any use of algorithms that cannot be bounded in execution time, full knowledge of the maximum execution frequency and worst-case execution times of every real-time task in the workload, and very conservative budgeting of memory and CPU resources to avoid any dependency on capabilities that cannot always be guaranteed (such as

memory cache hits, avoidance of pipeline hazards, and ability to execute instructions out of order). In general, developers of hard-real-time systems and, to an even greater degree, developers of safety-critical systems make completely different economic trade-offs than typical PERC developers.

These differences in development priorities are also manifest in the resources required for typical deployments. The implementation cost of the high-level abstractions provided by PERC, which include dynamic class loading, JIT compilation, and automatic real-time garbage collection, is measured in code size, system complexity, and throughput overhead. Typical PERC-based deployments are about 3 times as large and run at about $\frac{1}{3}$ the speed of comparable C programs.

Recognizing that different real-time developers have different requirements, our approach to real-time Java has been to offer developers a choice between several complementary realms of real-time Java technologies, each targeted to the needs of a slightly different audience. This is illustrated in Fig. 4. The most important attributes of this real-time Java framework are that (1) it allows strong compile-time enforced partitioning between the responsibilities of each software layer, and (2) it encourages disciplined high-performance sharing of information and cooperation of control between software running in each of the software layers. The PERC virtual machine that is the focus of this paper is an example of a firm-real-time Java product occupying the third ring from the center. The inner two rings are designed for hard-real-time software, offering performance, memory footprint, and real-time latency that is comparable to typical C development. Development of software targeted to these lower layers is much more difficult than development for PERC. These inner rings do not offer automatic garbage collection and lack the breadth of standard libraries and off-the-shelf 3rd party libraries that are readily available to J2SE programmers.

The PERC development environment has already demonstrated its value for development of firm-real-time systems. Even so, there are several areas in which we intend to further build upon the real-time programming abstractions offered by this platform. Several areas of future work are described below:

1. The hard-real-time technologies represented by the two inner circles of Figure have not yet been implemented. With funding support from the U.S. Navy, Aonix is currently implementing these hard-real-time Java technologies and Aonix intends to integrate these technologies within the PERC virtual machine.
2. A capability currently missing from the PERC run-time environment is the ability to asynchronously interrupt a particular thread's execution, as described in reference . As firm-real-time Java technologies expand to include broader classes of problems, we believe it will be important to add this capability.
3. In the highly complex, large, and dynamic applications that are best suited to the capabilities of firm-real-time Java technologies, it is very important to efficiently support independent development and straightforward integration of software components. In the domain of real-time systems, integration of independently developed components is especially challenging because new components often introduce increased competition for shared memory and CPU resources that are essential to compliance with real-time constraints. An important planned enhancement to firm-real-time Java technologies involves the ability to establish time and

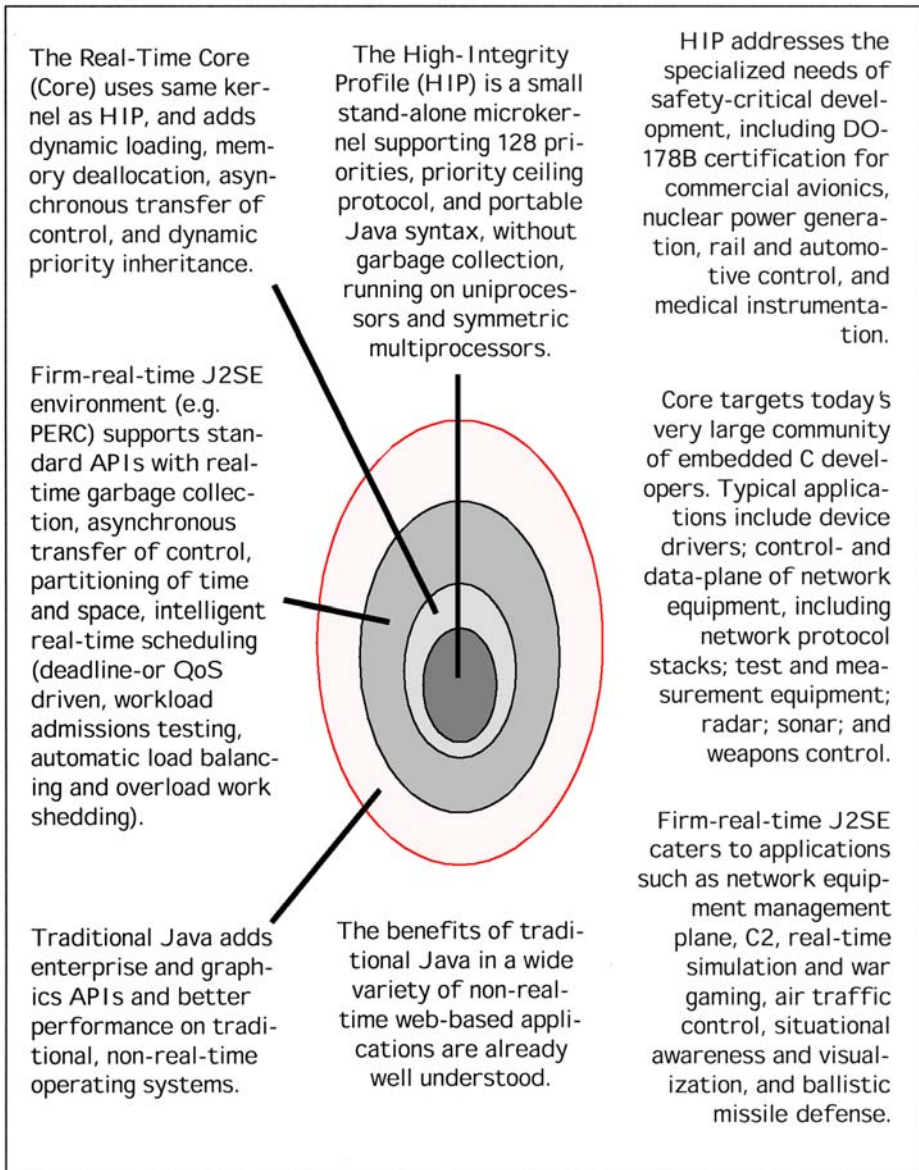


Fig. 4. A Hierarchy of Real-Time Java Technologies

space partitions for particular real-time software components. For example, on a particular system, component **A** may be assigned a partition of 1.5 Mbytes of heap memory and 15% of the total CPU time. Framework support for determining appropriate resource budgets for a given platform and enforcing these resource partitions is critical to supporting composability of large-scale firm-real-time software systems.

4. Schedulers for the hard-real-time domain are fairly simple because there are few if any judgment calls to be made by the scheduler. Almost always, considerable advance preparation has been invested to make sure that there will never exist a situation in which the tasks cannot be scheduled to meet all real-time constraints. In contrast, soft-real-time and firm-real-time schedulers are much more complicated. Generally, it is not known exactly how much time each task will require to execute or even how frequently each task will be triggered to execute. There is no guarantee that sufficient resources exist to always guarantee that all tasks will always meet all of their deadlines, and the question of how to deal with overload situations is quite complex. When overloads arise, should the scheduler insist on completing all tasks so that every task runs late, or should it selectively prune certain less important tasks from its active set to make sure the others continue to run on schedule? Would it be possible to force certain tasks to run with degraded service quality by simply decreasing their CPU resource budgets? Under overload situations, how do I decide which tasks are most deserving of oversubscribed CPU time? And how do I withhold this precious resource from those tasks that are less deserving? These sorts of questions make the real-time scheduling of firm-real-time systems much more challenging than that of hard-real-time applications. Adding an appropriate firm-real-time scheduling framework on top of firm-real-time Java technologies is another area of important future work.

5 Summary

“Real Time” means different things to different people. Each developer community makes different trade-offs between ease of development, the costs of porting and maintenance, memory footprint and performance expectations, and real-time predictability. A real-time implementation of the J2SE standard is well suited to real-time developers who are dealing with relatively large systems comprised of tens to hundreds of thousands of lines of code with tens of megabytes of RAM and firm-real-time deadlines measured in milliseconds. Other real-time Java technologies are better suited to applications that require hard-real-time, low-level control of hardware devices, safety certification, or compliance with severe footprint and/or demanding CPU performance expectations.

References

1. Gilder, G. “Will Java Break Windows?” *Forbes* ASAP, Aug. 25, 1997.
2. “Real-Time and Embedded Systems Forum”, Open Group Slide Presentation. Available at <http://www.opengroup.org/rtforum/info/slides.pdf>.
3. Nilsen, K., Klein, A. “Issues in the Design and Implementation of Efficient Interfaces Between Hard and Soft Real-Time Java Components”, Workshop on Java Technologies for Real-Time and Embedded Systems, Springer Lecture Notes in Computer Science. Catania, Sicily, Italy. Nov. 2003.
4. Zukowski, J. “Mastering Java 2, J2SE 1.4”, Sybex. 928 pages. April 2002.
5. Keogh, J. “J2EE: The Complete Reference”, McGraw-Hill Osborne Media. 904 pages. Sept. 6, 2002.

6. Keogh, J. "J2ME: The Complete Reference", McGraw-Hill Osborne Media. 768 pages. Feb. 27, 2003.
7. Bollella, G. et al. "The Real-Time Specification for Java", Addison-Wesley Publishing Company. 195 pages. Jan. 15, 2000.
8. "PERC 4.0 User Manual", Aonix, Inc., 308 pages, Nov. 14, 2002.
9. Nilsen, K., Mitra, S., Lee, S. "Method for Efficient Soft Real-Time Execution of Portable Byte Code Computer Programs", U. S. Patent No. 6,081, 665. 89 claims. June 27, 2000.
10. "XATA Unveils XATAnet Wireless Truck Information Software", ITS America News (www.itsa.org), January 3, 2001.
11. Nilsen, K., Mitra S., Sankaranarayanan, S., Thanuvan, V., "Asynchronous Java Exception Handling in a Real-Time Context", IEEE Workshop on Programming Languages for Real-Time Industrial Applications, Madrid, Spain. December 1998.

Container Model Based on RTSJ Services

Ruth Tolosa¹, José P. Mayo¹, Miguel A. de Miguel¹,
M. Teresa Higuera-Toledano², and Alejandro Alonso¹

¹ Department of Telematics Engineering, Technical University of Madrid.

² Faculty of Computer Science, Complutense University of Madrid,
Ciudad Universitaria, 28040 Madrid Spain
mmiguel@dit.upm.es

Abstract. The container is a basic structure of J2EE used to reduce the complexity of clients and applicative services. Component-based infrastructures use this type of structures and provide support for the development and execution of component-based systems. However, they have limitations in their application in real-time and reliable systems, because they neither integrate facilities to support these types of problems nor include services of predictability and dependability. RTSJ is a basic framework that includes the basic services for the construction of Java real-time systems. The combination of both Java approaches (Containers and RTSJ) is a good solution to reduce the complexity of real-time Java programs. The integration requires the adaptation of both approaches. In this paper we introduce a new model of component container that integrate the RTSJ services based on a resource reservation model.

1 Introduction

Currently, the development and implementation of real-time systems requires the detailed comprehension of some complex techniques (e.g. scheduling analysis techniques, real-time memory management), and the languages or APIs that support these concepts (e.g. RTSJ [2] and RTCore [10]). The integration of these APIs and techniques with application specific problems increases the complexity of design and implementations of applications and their maintainability.

J2EE uses middle-tier server solutions to reduce the complexity of clients and application services. EJB (Enterprise Java Beans) container is a basic structure to support the J2EE middle-tier architecture [5]. The container is a runtime environment that controls the business implementations and provides them with important system-level services. Since developers do not have to develop these services, they are free to concentrate on the application methods. The containers support the common technical solutions, and isolate dependencies of specific implementations. This approach avoids the problems of incorrect use of RTSJ APIs and simplifies the component. The system-level services that integrate current models of containers do not include real-time facilities such as RTSJ services. Current models of EJB are supported by services such as transaction, persistence, and security.

The model of container that introduces Section 2 and the basic services of RTSJ define a component model equivalent to other EJB object types (session, message, and entity), which address other types of problems. This new type of component gives support to represent conversational interactions that require limited response times and resource consumption. In this paper we introduce some solutions for the integration of RTSJ (Real-Time Specification for Java) APIs in container infrastructures. In this integration the container controls two basic resources that supports RTSJ: CPU and memory. The management of these resources in the container isolates in the component container the problems of concurrency, memory management, predictability and synchronization, which support RTSJ. In some previous works we have designed container models that provide predictability of methods invocations based on network reservation and admission control services [6,7]. This paper includes a container model that executes components in local mode (the system executes in mono-processor mode, as RTSJ) and we use the RTSJ resource services to improve the predictability of application components.

The Section 2 includes the general properties of this component model, Section 3 introduces some practical solution to support this model with RTSJ, Section 4 includes some practical results, Section 5 includes the related work and Section 6 includes the conclusions.

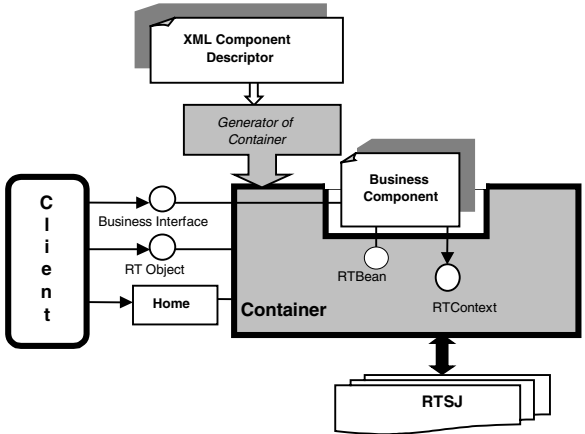


Fig. 1. RTC component model

2 Real-Time Component Model

The *resource-consuming component* is a processing entity that includes a group of concurrent units of execution, which cooperate in the execution of a certain activity and share common budgets. The *budget* is an assigned and guaranteed share of a certain resource. A component has associated: i) *facets* (interfaces provided and synchronously or asynchronously used by clients), and ii) *receptacles* (interfaces synchronously and asynchronously used by this component). A component can

negotiate the arrival-pattern and latency for its facets and receptacles. The process of negotiation establishes the temporal properties of component execution. The container implements the negotiation associated to the facets; the container negotiates that there are resources available and reserves the resources. Specific services included in the component's facet support the negotiation facilities and the container intercepts the invocation of these services. The negotiations are expressed in terms of the quality attributes that define the arrival-patterns and latencies of method invocations and memory usage that support the component.

Figure 1 includes the main blocks associated to the component and its development process. Home interface and the *Business Interface* are structures defined in EJB. The rest of block are inspired in EJB model, but we adapt them to real-time and RTSJ domains.

1. **External Interfaces.** The external interface of the component includes the *Business Interface* that identifies the business methods of the interface, the Home interface that includes the factory methods of the component, and the *RTObject* interface that is a common interface for all RTC (Real-Time Component) components. *RTObject* includes methods for the negotiation and other system services.

2. **Component Descriptor.** This XML file includes the identification of the component type and its attribute values. RTC includes attributes for the identification of the type of reservation that is used and the resources required in the component in general and in its methods specifically, the maximum number of references that the component can support, maximum number of concurrent invocations supported, scale of times used in time expressions, and execution times for applicative methods.

3. **Container.** The container supports the type of component specified in the *Component Descriptor*, for the *Business Interface*. The container implementations use RTSJ to guarantee the allocation of resources for the component, it manages the concurrency in the component, and uses RTSJ for the evaluation of the response time negotiated with clients. The *Generator of Container* has as inputs the component descriptor and the *Business Interface*, and generates automatically the container of this component. The container intercepts any method call to the component; it delegates the applicative methods, but during the execution it and the basic services monitor the method execution.

4. **RTContext.** This interface includes the internal services provided by the container to the business component. It provides methods for the dynamic specification of resources required (memory and execution times) during the component construction, provides information about the CPU available for the current budget, methods that evaluate the proximity of the deadline, methods that return the internal memory available in the component, and other general component services.

5. **RTBean.** The applicative component must implement the *RTBean* interface. In this solution the business component cannot customize the negotiation process as in [6], the component descriptor includes all the information to configure the negotiation. This interface includes methods for the notification of new reservations that can affect the execution of this component.

6. **Home.** The home is a component factory that creates component instances and returns component references. Each component instance can have associated a set of references and a set of clients can use the same reference to communicate with the same component instance.

2.1 Model of Method Invocation, Resource Reservation, and Negotiation

The interface `RTObject` includes a set of services for the negotiation with its clients the arrival-pattern invocation that the component can support (e.g. periodic, aperiodic, group-server), the specific parameters of each pattern (e.g. minimum period, maximum number of concurrent executions), and the allowed latency (e.g. deadline) for the responses. The arrival-pattern is a constraint that the client must achieve, and the latencies are the constraints that the component achieves, when the negotiation contract is done. The container intercepts these invocations and uses the RTSJ services to make the CPU reservation and detect the admission control problems. The containers support two types of reservations that depend on the method to describe the cost of execution in the component. i) The component descriptor includes the execution cost for each business method, and the arrival pattern includes the identification of methods that will be requested. ii) The client only specifies the percent of CPU that the component must reserve for any invocation associated to this reservation. In both cases, the container computes the budget, uses RTSJ services to make the reservation and to evaluate the new schedulability configuration.

The model of reservation could be based on several programming concepts. Three examples are: i) the negotiation process reserves resources for the specific client that makes the negotiation, ii) the negotiation affects to all clients of a component instance, and all clients share the same reservation, and iii) the reservation is associated to component references, and all clients that share the same reference, share the reservations, but different references have different reservations. The `Home` interface includes methods for the construction of new components and to get references to components created. We use the third solution, because it can support the other solutions. In this solution, the interface `Home` creates the references and initializes its reservation (initially they have a best-effort reservation). The container associates a pool of threads to the reference (in some cases this pool includes a single thread, the number and type of thread is included in the component descriptor). A multiple thread configuration allows the concurrent execution of invocations for the same reference. In this case, the reservation is a *group reservation* that limits the CPU consumption of the thread group. The resource manager reloads the cost for the execution for all threads every period. Depending on the number of threads, the method invocation can be blocked because there is no thread available, and it will stay blocked until the end of one method invocation for this reference.

2.2 Model of Memory Management

In our memory model the component has associated two types of memory spaces. One space support the instances handled inside the component (*internal memory*), this space is configured when the component is created. The second set of spaces (*external memory*) support the interchange of information (object instances of method parameters and return object values) with the component's client. The component description includes the default type of *internal memory* that the home factory uses, when the constructor do not specify the memory type. The default external memory is the memory active when the method invocation occurs. `RTObject` includes methods for the execution of methods in specific *external memories*. *Internal* and *external*

memories must respect the lifetime rules of RTSJ scoped references, and the lifetime of the component (and the *internal memory*) must be shorter than the lifetime of *external memories* in use.

The container updates the active memory in the method invocation and at the end of the method execution reestablishes the memory active in the invocation. The interface `RTContext` includes operations to create the return instances in the *external memory*. The component must not create references from the external objects to the *internal memory*.

The component descriptor includes information about the maximum memory allocation that have associated each method. This information is reused to compute the allocation time of threads and the allocation time is submitted to admission control. This model supports the local execution of components. This model supports the local execution of components. If the execution were distributed, the unmarshal code could instantiate the serialized objects in the *internal memory*.

2.3 Model of Synchronization

The sequence of execution of a method invocation can be blocked because of several reasons until the end of the method invocation. Different types of reasons are:

1. **Synchronized method.** This type of method avoids the concurrent executions inside the component. Different clients, with the same or different references, cannot execute concurrently the same or different *synchronized* methods for the same instance.
2. **Multiple invocations for the same reference.** A reference classified *periodic* or *aperiodic* has associated a single thread to serve the method invocations. A reference classified *group-server* has associated a limited number of threads. Concurrent invocations for the same reference *periodic* are not allowed. But several clients can use the same *group-server* reference simultaneously. If the number of clients (c) that execute concurrent invocations is more than the number of threads in the group (t), $c-t$ clients will be blocked until the end of method invocations.
3. **Container synchronizations.** The container uses some synchronization structures to maintain the consistency of references and other internal objects. For example to detect that a reference in use is not removed. The execution times for the operations that handle these structures are very short, but priority inversions may extend the blocking time.

3 Implementation of RTC Based on RTSJ

RTSJ includes services for the implementation of model of container introduced in Section 2. The basic services provide support for: i) the resource reservation management (CPU and memory), ii) synchronization and concurrency in the component model, iii) patterns of method invocation, iv) limitation of blocking times and response times. These services make the component response time predictable, and limit the resource consumption for the component.

3.1 Invocation Sequence, Resource Reservation, Admission Control

The predictability of response time of components requires contracting the temporal distribution of method invocation from clients. Depending on the type of invocation pattern and the temporal parameters, we must do the resource reservation. In Section 2 we consider three types of invocations (periodic, aperiodic, group-server) with specific parameters. RTSJ includes classes for the specification of release of threads (`PeriodicParameters`, `AperiodicParameters`, `SporadicParameters`) and the class `ProcessingGroupParameters` can group schedulable objects. The container creates pools of `RealtimeThreads` that serve the invocations for the references. The release parameters of threads depend on the type of invocation release and the temporal parameters that include the negotiation protocols.

In the negotiation process, the new `RealtimeThreads` are included as feasible in the schedulability analysis and the class `PriorityScheduler` returns the results for the admission control. The scheduling analysis takes into account the memory management information. The component descriptor includes the worst-case execution times for each application method. It will be used as cost parameter of constructor of classes

`PeriodicParameters`, `AperiodicParameters`, `SporadicParameters`.

Problems: Class `MemoryArea` provides methods for the computation of memory remaining in a memory block, but RTSJ does not includes services for the evaluation of CPU available for current budget. This reduces the types of negotiation services that the container can implement (the container can not negotiate based on the amounts of CPU non-reserved, or based on the CPU reservations not consumed). This requires specific resource management services implemented in lowest scheduling levels, or some of hooks in the context switch to compute the CPU consumed.

3.2 Component Memory Management

RTSJ memory areas that support the *internal memory* are `ImmortalMemory` and `LTMemory`. The structure of the component model, based on an interceptor design pattern, allows the interception of all invocations, and the container updates the active memory areas before the business method execution and reestablishes the active memory at the end of the execution. The component factory that implements the container for the instantiation creates a new `LTMemory` instance or configures the `ImmortalMemory` as *internal memory* and this do not change during the execution of the component.

The memory schema allows the reference to the external objects that represent the input parameters, during the execution of the component's methods. And the return objects are copied or created in the *external memory* to make references from the external memories. The containers implement the copy of return objects to external memory, when this is needed (if the *internal memory* is `ImmortalMemory` this is not needed). The component descriptor includes the *maximum allocation memory* (the maximum number of bytes that the method can allocate in one execution) for each method. This value and the temporal distribution of method invocations are used in

the computation of `allocationRate` parameter of `MemoryParameters`, which is given on the constructor of `RealtimeThread` and is used for the admission control.

3.3 Component Synchronizations and Blocking Times

The synchronizations for the access to the business component and for the race conditions in the container may block the execution of the invocation sequences. The synchronization protocols that support RTSJ (priority inheritance and ceiling protocol) limit the blocking times and avoid the inversion of priority. Classes `PriorityCeilingEmulation` and `PriorityInheritance` support the protocols and we can compute the worst-case blocking times. The container includes the information about the temporal distribution of invocations and the protocols that are used in the synchronizations.

Problems: RTSJ does not include services for the computation of blocking times or their impact in the feasibility analysis. Classes `RealtimeThread`, `Scheduler`, and `ReleaseParameters` includes information of temporal distribution of threads execution, but they and their subclasses do not include associations with synchronization classes. New classes that extend `Scheduler` or `PriorityScheduler` and `ReleaseParameters` can include new methods to define association with synchronization, and the new `Scheduler` class can take into account the blocking time in the feasibility analysis. Section 2.3 includes a type of synchronization (*multiple invocations for the same reference*) that cannot be computed as blocking time of single data resource as the rest of blocking times. The thread pool is an example of multiple instance data resource. This type of blocking requires specific algorithms for the computation of blocking times; [3] includes algorithms to compute the blocking time in multi-unit object protocols.

4 Practical Results

The execution results that we are going to introduce are based on an implementation of component model introduced in Section 2. This implementation does not use RTSJ but uses the services of a *Resource Manager* [8], for the reservation of CPU. The component, clients and resource manager execute in the operating system pSoS, and the hardware is a CPU for multimedia applications (TriMedia TM1000 card). The *Resource Manager* executes in pSoS OS and provides services for monitoring execution times, control of budgets and admission control. The *Resource Manager* assigns the budgets to clusters of tasks and controls the consumption and reload of budgets. The *Resource Manager* uses pSoS to schedule and monitor the execution of tasks. It includes services for the reservation of CPU and it manages the priority of tasks. It decomposes the priorities in two bands. The tasks that have not consumed its priority budget execute in the high priority band, and when they consume their reservation execute in the low priority band with a best-effort behavior.

The implementation of application interfaces is a set of synthetic operations with fixed CPU consumption (in the examples the components include three methods with execution times of 100, 150 and 200 milliseconds). The containers make the reservation of CPU as response to negotiation request of clients. In following scenarios, the clients make as much invocations as possible. The clients execute in the low priority band. We are going to introduce two execution scenarios: basic reservation operations, and references with cluster of tasks and multiple clients.

4.1 Basic Reservation Operations

This execution includes one component instance and three clients. The clients make the invocation of methods 0 and 1, and negotiate the frequency of invocation of these methods or the bare percent of CPU reservation. The sequence of operations for the clients is included in next Table.

Table 1. Reservation of CPU for Scenario 1

Time	Client	Reference	Reservation
0	0	0	No reservation
0	1	1	50% of CPU
180	2	2	method 0, 2 times per second (20% CPU)
480	2	2	method 0, 4 times per second (40% CPU)
780	1	1	remove the reference 1
1080	0	0	method 1, 4 times per second (60% CPU)
1380	2	2	remove the reference 2
1680	0	0	method 1, 2 times per second (30% CPU)
1980	0	0	remove the references 0 and 2

The container creates a cluster of tasks for each reference. During the negotiation process, it makes the reservation of CPU for each cluster. Figure 2 includes the monitoring result for each cluster that provide the *Resource Manager*. The Figure includes the amount of CPU that they use of their reservation. Reference 0 has a best-effort behavior until it makes a reservation. Reference 1 removes its reservation and therefore its budget. Both references also make other modifications of their reservations. The clusters created for each reference disappear when the client removes the reference.

Figure 3 includes the response time of method invocation for the different references. The unit of axis y is 10 milliseconds. Figure 3 (a) includes the reference 0 that vary its response time until it makes a reservation (instant 1080). The response time and its variance are bigger when the total reservation of references 1 and 2 is the 70% and 90%. The reference 1 in Figure 3 (a) has response time less than 200 milliseconds for all the execution (because it executes with a base reservation of 50% of CPU). In Figure 3 (b) we can see two response times higher than 100000 milliseconds, when the reference 2 removes its reservation (because of the preemption of reference 0). And between the instants 1500 and 2000 it has no reservation, but it reduces its response times when the reference 0 reduces the reservation.

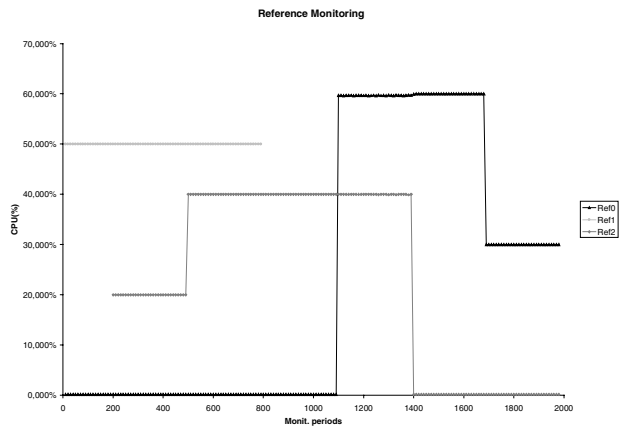


Fig. 2. Reservation of CPU for Scenario 1

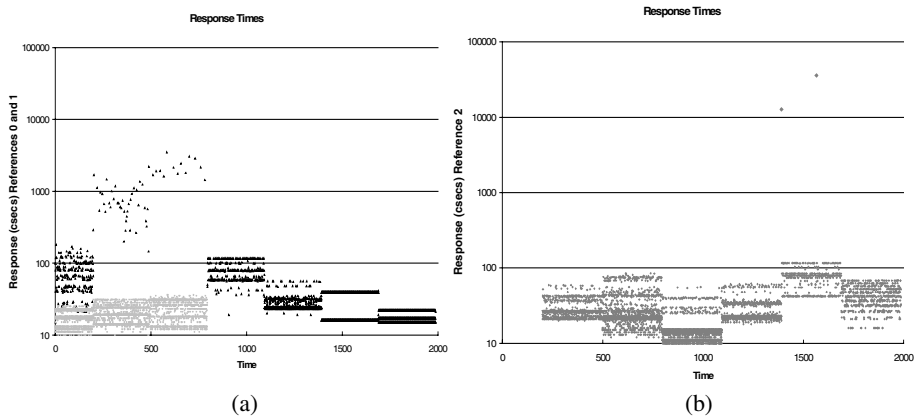


Fig. 3. Response Times for Scenario 1

4.2 References with Multiple Clients and Clusters

This scenario creates three references, but there are three clients for each reference. The Table 2 includes the sequence of reservation. All clients compete to make their invocations, and the different reservations produce different response times, for the different groups of clients.

Figure 4 includes the response time for the three references. The unit of axis y is 10 milliseconds. Figure 4 (a) includes the response times of reference 0. From instant 300 until instant 1620 the clients that use the reference 0 must compete with the clients that use the reference 1 and 2. Because reference 0 has no reservation and during this interval and the CPU is busy, the number of executions is reduced and their response time is high. In the Figure 4 (b) we can see the different of response time for the references 1 and 2, when the reservation of CPU is 20% and 40%.

Table 2. Reservation of CPU for Scenario 2

Time	Client	Reference	Reservation
0	0, 1, 2	0	No reservation
300	3, 4, 5	1	20% of CPU
600	6, 7, 8	2	method 0, 2 times per second (20% CPU)
960	3, 4, 5	1	40% CPU
960	6, 7, 8	2	method 0, 4 times per second (40% CPU)
1320	3, 4, 5	1	remove the reference 1
1620	6, 7, 8	2	remove the reference 2
1980	0, 1, 2	0	remove the reference 0

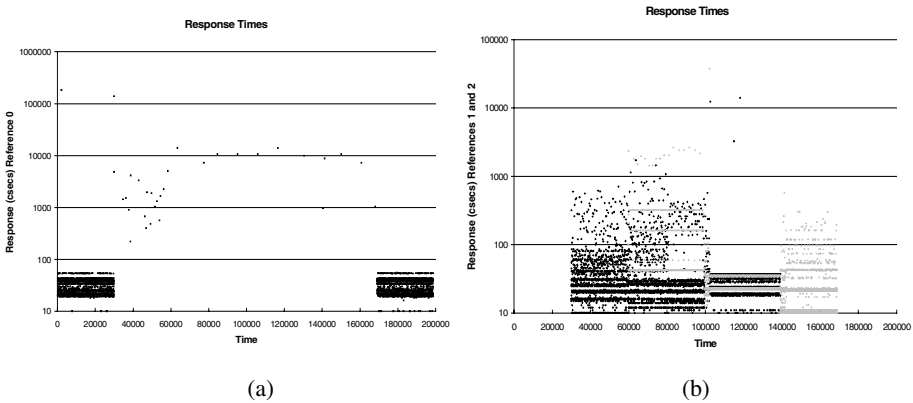


Fig. 4. Response Times for Scenario 2

5 Related Work

The integration of QoS in component infrastructures is a subject that has a very short history. Most of Component infrastructure standards (EJB 2.0, CCM and .NET) are very recent, and their integration with QoS and Real-Time facilities requires some basic practical improvements (e.g. CCM does not have industrial implementations yet, and open source implementations of .NET has started to appear last months).

Some proposals study the integration of QoS facilities in component models such as CCM (CORBA Component Model) [13,9]. The OMG is currently analyzing propose an RFP (Request for Proposal) for the extension of CCM with basic QoS facilities. The proposal by Wang et al [13] pays special attention to the QoS-enabled location transparency, reflective configuration of component server and container, and the strategies to reconfigure the component server. COACH IST [4] project includes some activities for the integration of QoS facilities in CCM standard. These extensions define generic interfaces to allow negotiation of QoS characteristics between supplier and consumer CORBA components.

Lusceta [1] is a component model (it is not based on industrial component infrastructures) environment based on formal techniques, which can be simulated and analyzed. *Lusceta* provides support for the specification of QoS management, which

can be used to synthesize (dynamic) QoS management components. The execution framework is a general QoS-aware reflective middleware. Another component modeling environment is presented in [11]. It proposes solutions for the description of component architectures and for evaluation of response times. This is an architectural environment not supported by execution environments.

[6,7] introduces a solution for the integration of QoS basic services, such as resource reservation and negotiation, in EJB (Enterprise Java Beans). The EJB containers implement some basic negotiation algorithms and isolate the business components from reservation services. The negotiation algorithms implement some basic adaptation process based on the renegotiation of resources and renegotiation with other components.

Schantz et al [12] describe how priority and reservation-based OS and network QoS management mechanisms can be coupled with standards-based, off-the-shelf distributed object computing middleware to better support dynamic distributed real-time applications with end-to-end real-time requirements. They compare two solutions based on priorities and resource reservation for CPU and bandwidth. The reservation solution is based on RSVP and TimeSys Linux RTOS, and the priority-oriented solution is based on Diffserv and TimeSys Linux RTOS.

6 Summary and Discussion

RTSJ provides basic services to support the component model that introduces Section 2. This component model provides high level facilities to make Java components time predictable. The container structure reduces the complexity of applications, and its configured based on attributes. These attributes characterize the temporal behavior and resource consumption of components, and this solution avoids the detailed comprehension of RTSJ complex APIs.

RTSJ can be improved to support models of predictability based on resource reservation. RTSJ include basic services for the evaluation of feasibility of response times, but do not provide information about amount of CPU available, occupation of CPU depending on amounts available, redistribution of non-used CPU (worst case CPU occupation is less than 100%, or the threads do not consume their worst case execution times). Another improvement is the integration of blocking time evaluations in the feasibility schemas. RTSJ includes classes that support time predictable synchronization protocols, but RTSJ do not provide information about blocking times, and there is not associations between these classes and deadline feasibility analysis.

References

1. L. Blair, G. Blair, A. Andersen and T. Jones. "Formal Support for Dynamic QoS Management in the Development of Open Component-based Distributed Systems". *IEE Proceedings Software*. Vol. 148 No. 3. (June 2001).
2. G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Sun Microsystems, 2000.

3. M. Chen and K. Lin. "A Priority Ceiling Protocol for Multiple-Instance Resource". Proc. of IEEE Real-Time Systems Symposium, (1991).
4. Coach IST Project. <http://coach.objectweb.org>
5. L. DeMichiel, L. Yalinalp, and S. Krishnan. *Java 2 Platform Enterprise Edition Specifications*, v2.0. Sun Microsystems, 1999.
6. M. de Miguel, J. Ruiz and M. García, "QoS-Aware Component Frameworks", Proc. International Workshop on Quality of Service, (May 2002).
7. M. de Miguel "Integration of QoS Facilities into Component Container Architectures", Proc. 5th IEEE Object-Oriented Real-Time Distributed Computing. IEEE Computer Society, (May 2002).
8. Marisol García-Valls, Alejandro Alonso, José F. Ruiz, and Ángel Groba "An Architecture for a Quality of Service Resource Manager Middleware for Flexible Multimedia Embedded Systems", Proc. International Workshop on Software Engineering and Middleware - SEM 2002. Orlando, Florida (2002)
9. Gokhale, D. Schmidt, B. Natarajan and N. Wang, "Applying Model-Integrated Computing to Component Middleware and Enterprise Applications", *The Communications of the ACM Special Issue on Enterprise Components, Service and Business Rules*, Vol. 45, No. 10, (October 2002)
10. J. Consortium Inc. "Core Real-Time Extensions for the Java Platform". Technical Report. New-Monics Inc. <http://www.j-consortium.org>. (2000).
11. U. Rasthofer and F. Bellosa. "An Approach to Component-based Software Engineering for Distributed Real-Time Systems". Proc. SCI 2000 Invited Session on Generative and Component-based Software Engineering. IIS (2000).
12. R. Schantz, J. Loyall, C. Rodrigues, D. Schmidt, Y. Krishnamurthy and I. Pyarali. "Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware". Proc. Middleware 2003. (June 2003).
13. N. Wang, D. Schmidt, M. Kircher, and. K. Parameswaran. "Adaptative and Reflective Middleware for QoS-Enabled CCM Applications". *IEEE Distributed Systems Online* Vol 2 No. 5. (July 2001).

Reference Objects for RTSJ Memory Areas

Andrew Borg and Andy J. Wellings

University of York, UK,
email{aborg,andy}@cs.york.ac.uk,
<http://www.cs.york.ac.uk/rts>

Abstract. The RTSJ introduces a memory model that includes immortal and scoped memory areas that are not subject to garbage collection. Associated with scoped memory areas are rules that restrict reference assignments to guarantee dangling pointers can not occur. Java includes special references which have different semantics than regular references. Encapsulated by *reference objects*, these references are treated uniquely way by the garbage collector. In particular, they do not stop their references from being marked for finalization, finalized and reclaimed.

This paper makes four contributions. We first propose semantics for reference objects used in the RTSJ. We then propose a new *scoped reference object* that allows access to objects where this would be disallowed by the RTSJ scoping rules. Next, we show how reference object reachability changes can be processed using RTSJ asynchronous events. Finally, we show how thread groups, unavailable to RTSJ *RealtimeThreads* can be implemented safely using scoped reference objects.

1 Introduction

Java has no mechanism for reclaiming the memory used by unreachable objects. Consequently, an implementation must provide some form of garbage collection. However, real-time threads may fail to meet their timing requirements if interrupted by the garbage collector. While an incremental garbage collector [1,2] can provide bounds for the latencies a thread may experience due to garbage collection, there is generally a lack of confidence in these algorithms. The Real-Time Specification for Java (RTSJ) [3] introduces an alternative memory management model based on the concept of memory regions [4,5,6]. Besides the traditional heap, immortal and scoped memory areas are defined which are not garbage collected. Objects created in immortal memory live for the duration of the application while those created in scoped memory areas are reclaimed when there are no *Schedulable* objects active in that area. The `java.lang.ref` package defines classes that allow the programmer a limited degree of interaction with the garbage collector [7]. In particular, three classes are defined that allow the creation of *reference objects*. A program may use a reference object to maintain a reference to some other object in such a way that the latter object may still be garbage collected. A program may also arrange to be notified some time after the collector has determined that the reachability of a given object has changed.

This paper explores how reference objects may be used in the RTSJ and proposes a new reference object for scoped objects.

The rest of this paper is set out as follows: Section 2 introduces the `java.lang.ref` package. Section 3 proposes semantics for applications using these classes in the RTSJ. We then propose an extension to the RTSJ in Section 4 inspired by reference objects. This extension defines a new reference object that allows the creation of references to scoped objects in a way that is currently disallowed by the RTSJ reference rules. Section 5 illustrates how RTSJ asynchronous events can be used to provide notification of reachability change of a reference object. Section 6 shows how our extension can be used to implement thread groups for `RealtimeThreads`. Finally, Section 7 concludes.

2 Reference Objects

The assignment of an existing or new object in Java creates a reference called a *Strong Reference*. A strongly referenced object can not be garbage collected, that is marked for finalization, finalized and its memory reclaimed. The three reference objects provided in the `java.lang.ref` package are *Soft References*, *Weak References* and *Phantom References*. These are created by instantiation of the `SoftReference`, `WeakReference` and `PhantomReference` classes respectively.

2.1 Using Reference Objects in Java

The constructors for reference object classes take a reference to some object as a parameter for which the required reference is created. For example in the code:

```
Object x = new Object();
WeakReference w=new WeakReference(x);
```

`x` strongly references the newly created object while `w` is a weak reference to that object. The object referred to by a reference object is called the *referent*. Here, `x` is the referent of `w`. The reference object classes extend the abstract class `Reference` that provides the basic functionality for all reference types.

The `java.lang.ref` package contains a field called `referent` that provides the hook used by the application into the garbage collector. Assignments to this field are treated specially by the garbage collector. A set of methods in this package provide the interface for working with reference objects. The `get()` method returns a (strong) reference to the referent if the object has not already been marked for finalization by the garbage collector and `null` otherwise. (As we shall see later, phantom references are exceptions in that they always return `null` from `get()`). The `clear()` method clears the reference object; that is it sets the `referent` field to `null`. When any reference object is cleared, `get()` returns `null` for all reference object types. The garbage collector may free the memory occupied by referents once all the associated reference objects are cleared and if the object is not strongly reachable. Note that reference objects are immutable, that is there is no `set()` method to set the `referent` field.

Java defines the *reachability* of an object based on the type of reference an application holds to that object. An object is *strongly reachable* if it can be reached without traversing any reference object. It is *softly reachable* if it is not strongly reachable but can be reached by traversing a soft reference. It is *weakly reachable* if it can be reached by traversing a weak reference but it is not strongly or softly reachable. Finally, it is *phantom reachable* if it can be reached by traversing a phantom reference but is not strongly, softly or weakly reachable. Otherwise, an object is said to be *unreachable*.

Reference objects allow for some limited degree of interaction with the garbage collector in two ways. Firstly, whereas objects that are strongly referenced are never finalized as long as there remains at least one strong reference in the heap, reference objects do not prevent their referents from being collected. Secondly, a referent object may register itself with a `ReferenceQueue` onto which it is enqueued when its reachability changes. Each reference object encapsulates different semantics that define when: (1) the referent of a reference object can be marked for finalization, (2) the reference object is cleared, (3) the reference object is enqueued on a registered reference queue and (4) the referent's memory is reclaimed. We next described each reference type in more detail.

Soft References. Soft references are cleared at the discretion of the garbage collector in response to memory demand. When low on memory, the garbage collector may choose to clear atomically all soft references to a softly reachable object and all soft references to any other softly-reachable objects from which that object is reachable through a chain of strong references. The referent is marked for finalization and the referent's memory may be reclaimed when finalization is complete. At that point or at a later time, the cleared reference objects registered with a reference queue are enqueued. Soft references are used for implementing memory sensitive caches.

Weak References. When the garbage collector determines that an object is weakly reachable, it will atomically clear all weak references to that object and all weak references to any other weakly reachable objects from which that object is reachable through a chain of strong and soft references. It will declare the referent finalizable and at the same time or at some later time it will enqueue those newly-cleared weak references that are registered with reference queues. As with soft references, since weak reference objects are cleared automatically by the garbage collector, the referent's memory may be reclaimed when finalization is complete. Weak references are used for implementing canonicalizing mappings.

Phantom References. Phantom references differ from soft and weak references in two key ways. Firstly, invoking `get()` on an instance of `PhantomReference` always returns `null`. Therefore, phantom references must be registered with a reference queue to be useful. Secondly, phantom references are not automatically cleared by the garbage collector as they are enqueued. At this stage, the referents

are only marked for finalization. An object that is reachable via phantom references will remain so until all such references are cleared or themselves become unreachable. Therefore, though finalized, the memory of a phantom reachable object is not reclaimed until all phantom references are cleared. For this reason, at the end of the post-finalization cleanup code, the program should call the *clear()* method on the phantom reference object to set the reference field of the reference object to **null**, allowing the garbage collector to reclaim the referent's memory. Phantom references are used for implementing pre-mortem cleanup actions in a more flexible way than is possible with finalization

3 Using Reference Objects in the RTSJ

We now explore the semantics we believe would be correct for an application using reference objects in the RTSJ. We discuss how the use of noheap threads that may interrupt the garbage collector places implementation concerns on two key virtual machine functions when using reference objects: clearing of reference objects and enqueueing on the registered reference queue. We also show how and when heap and noheap threads may use *get()* and process reference queues. We use the notation $A \rightarrow B$ to signify that the reference object in memory area A references the referent in memory area B.

Heap $\rightarrow$ Heap: The same reachability semantics apply as in regular Java and the use of *get()* and the clearing of references objects is also the same for all reference object types.

The reference queue to which a reference object in heap is registered can be created in heap or immortal memory. In either case, processing of the reference queue should be done by a heap-using thread. Therefore, enqueueing of the reference object on a reference queue in immortal memory may leave the queue in an inconsistent state when interrupted by a noheap thread. Nevertheless, an implementation would probably still use an appropriate enqueue algorithm that leaves the queue in a consistent state at each step of the enqueue operation and throw a **MemoryAccessError** in the usual way if a reference object in heap is dequeued by a noheap thread.

Immortal $\rightarrow$ Heap: The semantics here are again the same as in regular Java for reachability, though the use of *get()* differs in compliance with noheap thread usage. A new situation arises here in which the garbage collector modifies immortal memory. This occurs when soft and weak reference objects are cleared by the virtual machine. Therefore, it is necessary to ensure that clearing reference objects does not incur penalties on noheap threads. Even if clearing is done atomically, this would incur priority inversion, albeit bounded, on the noheap thread by the garbage collector for the duration of the atomic operation. A better solution is therefore to automatically fail with a **MemoryAccessError** when calling *get()* from a noheap thread, even if the result of this would have been **null**. The latter solution means that reference objects can be safely left in a semi-cleared state as long as they are fully cleared before any other heap-using

thread executes. For cases requiring atomic clearing of multiple soft and weak references, both solutions are still valid as heap threads see an atomic clearing of the relevant reference objects.

Reference objects in immortal memory may be registered with reference queues in immortal or heap memory (of course the latter option is unavailable for noheap threads). Note that a noheap thread can interrupt the garbage collector's enqueueing thread at any point and require access to the reference queue. This requires the enqueue operation to be wait-free [8] as discussed in Section 5.3. Reference objects in immortal memory can always be enqueued onto a reference queue, even if they are unreachable as they are guaranteed not to have been collected.

Scoped $\rightarrow$ Heap: Again, all three types of reference objects may be created with the same reachability semantics. The rules for using *get()* and clearing reference objects are the same as for Immortal $\rightarrow$ Heap.

Reference objects point to the reference queues they are registered with. Once their reachability changes, it is the reference queue that points to the reference object. This implies that reference objects can only be registered with a reference queue in the same scoped memory area. Enqueueing of reference objects must be done as described for Immortal $\rightarrow$ Heap. Note that all referent objects created in scoped memory together with their reference queues are lost when the scoped memory area is removed. These semantics are the same as when a reference object is itself not strongly reachable and is garbage collected in standard Java.

{Heap, Immortal, Scoped} $\rightarrow$ Immortal: Fails for all reference types, throwing an exception. Immortal memory is never to be subject to garbage collection, meaning that objects are never to be marked for finalization. Though the garbage collector scans immortal memory and may therefore identify objects that are not strongly reachable, we can only assume that, in the most general case, the garbage collector algorithm considers all objects in immortal memory to be root objects and therefore strongly reachable [9].

3.1 References to Scoped Objects

It would be expected that the creation of reference objects to scoped referents would be governed by the same rules as those of the RTSJ. We show that this is not the case and some benefit can be derived by allowing phantom references to be created for referents in shorter-lived memory areas. The reachability semantics are different from regular Java as reachability of a referent now depends on the scoped memory reference count and not on the reachability of the referent. We study each reference type and identify how they may be used in this context.

Heap $\rightarrow$ Scoped: Referents of soft references are removed in response to memory requirements. However, even if a scoped memory region runs out of space, identifying softly reachable objects requires computation time. Doing this is contrary to the rationale for having non-garbage collected memory areas in the first place. It may be argued that weak references could be used but must be done

so with care as invoking *get()* would fail if the returned reference is assigned in a way that violates RTSJ rules. However, weakly reachable objects reachable by strong references from a cleared weak reference need to be cleared automatically and atomically by the garbage collector. These could exist in heap, immortal memory or the current and outer nested scopes. This would require immortal and scoped memory to be locked, potentially leading to noheap threads experiencing priority inversion due to the garbage collector. It is therefore more appropriate to only allow phantom references to be used to reference scoped objects as these are not cleared by the garbage collector. Clearing of the reference object by the application becomes redundant as the referents are reclaimed when the scoped area is cleared. Moreover at the time of enqueueing, the referent may have already been reclaimed. Therefore, it is not possible to determine whether premortem or post-mortem cleanup would take place.

The reference queue can be in either heap or immortal memory. The garbage collector must be able to identify whether the scoped area of a phantom reference has been cleared in order to enqueue that reference object. The rules for enqueueing in the virtual machine and processing of the queue by the application are the same as for $\text{Heap} \rightarrow \text{Heap}$.

Immortal $\rightarrow$ Scoped: As for $\text{Heap} \rightarrow \text{Scoped}$ above, we propose allowing only phantom references with the same reachability semantics. The rules for enqueueing and processing of the queue by the application are the same as for $\text{Immortal} \rightarrow \text{Heap}$.

Scoped $\rightarrow$ Scoped: There are four possible scoped memory locations for the referent of a scoped reference object. The first is when the referent is in an outer scope. The new reachability semantics of scoped reference objects makes this option useless. Reachability changes of the referent can only be detected when the scope memory holding that referent is cleared. As this point the reference object and reference queue would also have been cleared. The same holds for the second case where a referent's reference exists in the same scoped memory area. In the last two cases the referent is either in an inner scope or in a scoped memory area outside the current scope stack. Here we only allow phantom references for the same reasons given for $\text{Heap} \rightarrow \text{Scoped}$ references above.

As for $\text{Scoped} \rightarrow \text{Heap}$ references, reference queues can only be in the same scoped memory as the reference object. Enqueueing and processing of reference objects is the same as for $\text{Heap} \rightarrow \text{Scoped}$ references above.

4 Scoped Reference Objects

The use of reference objects described above is limited in the RTSJ, particularly when the referents exist in scoped memory. Reference objects for referents in immortal memory are discounted because of the permanent nature of objects therein. Scoped memory objects however are cleared, but the resultant interaction with the garbage collector discounts the use of soft and weak references here. While soft references remain discounted as these would change the semantics of

scoped regions, a new type of reference object similar to a weak reference and that allows access to scoped objects outside current RTSJ rules could be useful for application devolvement. In this section we propose a new reference object called a *scoped reference object* that achieves this.

4.1 Beyond the RTSJ Scoping Rules

The RTSJ requires the clearing of scoped memory regions to depend on the access of `Schedulables` to the scope. While immortal memory is never cleaned, scoped memory allows for a more coarse-grained memory reclamation mechanism than traditional garbage collection. Since clearing of scoped regions is not dependant on access of an object, any reference assignment to an object from a memory area that may outlive the owning scope of that object is disallowed. It would nevertheless be attractive for an application developer to be able to access scoped objects and execute their methods from longer lived memory areas.

A naïve solution to allow any object to reference a scoped object is to increment the reference count of the scoped memory region and all outer scopes whenever an object from heap, immortal, inner scope or external scoped memory creates a reference to that scope. If this is done then no dangling pointers can ever occur. This solution makes region lifetime dependent on references, removing the desirable course-grain clearing semantics of whole regions. Moreover, decrementing the reference count may depend on the garbage collector identifying unreachable objects in all memory areas. Objects in immortal and scoped memory can no longer be assumed to be strongly reachable for garbage collection purposes. Also, the garbage collector would need to modify scoped memory area reference counts, potentially leading to noheap threads having to wait for the garbage collector before knowing whether it is safe to enter a region. This failing solution highlights two properties that a correct solution should have and motivates the solution we propose next:

- The solution must not leave the lifetime of the scoped region dependant on the garbage collector but should still reflect the number of `Schedulable` objects currently executing in that region.
- Dangling pointers must not occur arbitrarily. Our exploration of soft and weak reference objects shows that it can be safe to hold a reference to an object that may be removed as long as the application knows this may occur.

4.2 The `ScopedObject` and `ScopedReference` Classes

We first introduce a new tagging interface called `ScopedObject`. Instances of classes that implement this class may be created in scoped memory and be referenced by special scoped reference objects. Scoped reference objects are proxy objects which hold a reference to the scoped object (the referent). We introduce the class `ScopedReference` that extends `Reference` and contains the static method `getProxy()` that returns such a proxy object. The application invokes

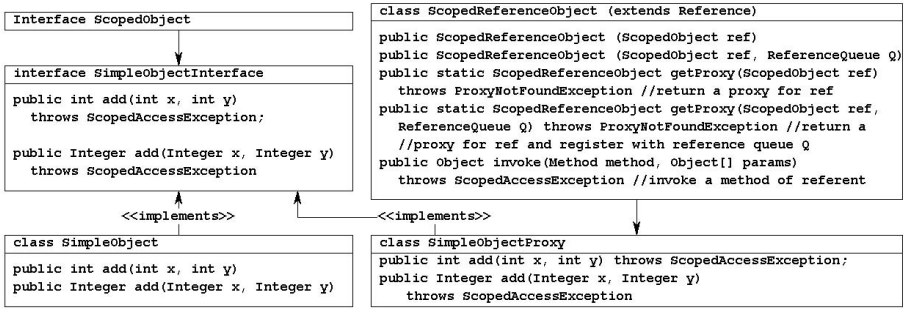


Fig. 1. Class Hierarchy when using Scoped Reference Objects

methods of the referent by invoking the methods on the proxy. If the virtual machine deems that the invocation may fail, an exception `ScopedAccessException` is thrown by the proxy. Using a proxy makes `get()` redundant and it is therefore overridden by `ScopedReference` to return `null`. Invoking `clear()` on a scoped reference object makes any subsequent invocation on the proxy fail.

We illustrate the class hierarchy created by scoped reference objects with an example as depicted in Figure 1. Consider a class `SimpleObject` that contains the methods `int add(int x, int y)` and `Integer add(Integer x, Integer y)` that return the sum of `x` and `y`. If an instance of `SimpleObject` is to be referenced by a scoped reference object, then a new interface `SimpleObjectInterface` that extends `ScopedObject` must be defined that describes all methods in `SimpleObject` that can be invoked from the reference object. Each method must throw the checked exception `ScopedAccessException`. A (possibly dynamic) proxy generator is used to generate proxy classes that are by convention given the name of the implementing class concatenated with the string “Proxy”. Hence, the proxy generator in this case generates a class `SimpleObjectProxy`. `SimpleObjectProxy` is made to extend `ScopedReference` and implement `SimpleObjectInterface` and is therefore type compatible with the referent.

When `getProxy()` is called by the application, the referent of type `SimpleObject` is passed as a parameter in the call. The result is an instance of `SimpleObjectProxy`. An invocation on this proxy calls the `invoke()` method in `ScopedReferenceObject` that attempts to invoke the associated method of the referent. The virtual machine is consulted to ensure that the invocation can take place. If the scoped memory region holding the object still exists, an attempt is made to increment the reference count for that scope and all outer scopes of the referent. If this fails, a `ScopedAccessException` is thrown. The working memory during the invocation is that of the reference object and not of the referent.

Scoped reference objects may be registered with reference queues and appended to the queue at some point after the reference count of the scoped memory holding the referent reaches 0. Typically this operation would be carried out at the time the scoped region is to be reclaimed and the finalizers of objects in

the scope are run. Just as there is no guarantee when regular reference objects are enqueued once deemed unreachable by the garbage collector, there is no guaranteed when scoped reference objects are enqueued. Each scoped memory area should maintain a list of scoped reference objects it needs to append to the reference queue once it is cleared. Scoped reference objects in heap can be assumed to always be strongly reachable until the referent's scope is cleared and, therefore, never subject to garbage collection.

Analysis of Scoped References Solution. The solution proposed above no longer depends on the garbage collector in any way. Reachability is no longer defined by reachability of the referent by the application but by the state of the referent's scope. Since the **referent** field may be holding an invalid reference, any invocation on that object may fail. Requiring the application to catch an exception if this occurs ensures integrity of the program. Therefore the two requirements described above are met. Scoped reference objects are different from regular reference objects in that the reference object is in fact the object that is invoked to achieve functionality of the referent.

Implementing Scoped Reference Objects. We sketch a simple implementation that highlights a number of important considerations. Firstly, when a new scoped reference object is created, the virtual machine must allow the referent to be set, even if this breaks the RTSJ rules. As the referent may be left dangling at any time, the garbage collector should not try to access the memory area that is internally pointed to by the **referent** field. Once a scoped reference object is successfully created, **invoke()** can be called by invoking a method of the proxy. **invoke()** could be implemented by calling some native method that finds the scope stack of the parent and attempts to climb it, atomically incrementing the reference count as it goes along. This process may be safely interrupted by a higher priority thread at any time. If at any time an attempt is made to enter a region with a reference count of 0, the native method is made to retreat back down the scope, decrementing the reference count of each region and finally throwing a **ScopedAccessException**. If the region holding the referent is reached, it may still be that, prior to the invocation being made, the region had been cleared. Therefore, the **referent** field held by the scoped reference object is invalid. To solve this problem, each scoped memory area has a *clear count* that is incremented every time it is cleared. The scoped reference object will save this count when it is created and the invocation will succeed only if the two are equal. On completion of the invocation, another native method is called to decrement the reference count of each region in the stack.

This implementation makes it possible to bound the time taken for the referent's method to begin execution by a function of the scope depth of the referent's owning scope. If the invocation fails, the scoped reference object should be cleared so that subsequent invocations can fail immediately. While it is possible to allow scoped memory regions to recall the scoped reference objects for which they hold referents, clearing those reference objects once the region is reclaimed,

the RTSJ does not guarantee when clearing will happen. This is why it is always necessary to work up from the bottom of the stack and not assume a non-cleared reference is a valid one.

5 Real-Time Reference Object Processing

Reference queues must be processed by an application in order to carry out the necessary logic when the reachability of the reference object changes. It is impossible to guarantee when a reference object is enqueued onto a reference queue due to the unpredictability of when the garbage collector runs or, for scoped reference objects, when the scope is cleared. However, the processing of reference objects can be done in a timely fashion using RTSJ *Schedulables*.

5.1 Using Reference Queues

The `ReferenceQueue` class provides three methods to extract reference objects from a reference queue: *poll()*, *remove()* and *remove(int timeout)*. The first method is a non-blocking call that returns `null` if no reference object is enqueued. The latter two methods are blocking calls, the latter of which waits for a specified number of milliseconds. An application would typically use *poll()* and *remove(int timeout)* if no dedicated thread is to be assigned to the processing of the reference queue. If a dedicated thread is to be assigned to reference queue processing, *remove()* is called in the processing loop, blocking until an object is available in the queue. A reference queue may be processed by any number of threads, each calling any of these three methods.

The timeout parameter for *remove()* does not have real-time characteristics as a direct repercussion of the absence of real-time functionality in Java. Typically, the timeout is implemented using a *wait(int timeout)* operation on a lock that must be obtained before any enqueue or dequeue operation on the queue instance. If a reference object is enqueued by calling *enqueue()*, *notify()* is invoked to wake up an arbitrary thread blocked on one of the *remove()* methods.

5.2 Processing Reachability Change in Real-Time

Having multiple threads to service the same reference queue using blocking calls makes predictability hard. There are two solutions to this problem. The first is to associate just one `RealtimeThread` with assigned scheduling and releases parameters to one or more reference queues. Assuming a periodic thread is used, *poll()* is invoked at every period, servicing one reference object from a selected queue each time. In this case, the worst case execution time would be the greatest of the worst case times to process any of the reference queue objects.

An more attractive solution for notifying an application of reachability change of a reference object is to make use of the RTSJ's asynchronous event handler functionality. However, as a result of the data-less association between events

and their handlers, unnecessary complexity is introduced unless only one handler is assigned to each reference object. Therefore, a more appropriate solution is to associate an event directly with a reference object, rather than with a reference queue. We define a new abstract class **RealtimeReference** that extends **Reference**. Constructors of subclasses of **RealtimeReference** may take an **AsyncEvent** instance that may be associated with a set of **AsyncEventHandlers** in the usual way. When the reference object is cleared, the event is fired. Associating a unique event to each reference object will allow the application to identify which reference object has been cleared.

5.3 Implementing Reference Queues for Real-Time

Even with the above two solutions, it may be necessary to allow multiple threads to access a reference queue. In fact, reference queues are objects that are somewhat unique in that they are accessed and mutated by both the virtual machine and the application. An implementation would typically use a lock for the queue on which threads synchronise before adding or removing items from the queue. In Section 3, it was noted that in some cases the enqueue operation could be interrupted by a noheap thread. In particular, this could lead to a situation in which a noheap thread may have to wait for the garbage collector's enqueueing thread to complete its operation, leading to priority inversion caused by the garbage collector. Although this inversion may be bounded, it is still undesirable. Therefore, it is best to implement the queue without synchronisation on locks. The RTSJ provides this functionality in the form of wait-free queues. In particular, the reference queue can be implemented as a **WaitFreeWriteQueue** with an unsynchronised and nonblocking *write()* and a synchronised and blocking *read()*. The enqueue thread will simply fail to enqueue if interrupted by a noheap thread and attempt to enqueue again some time later.

6 ThreadGroups for RealtimeThreads

Every Java thread an application creates is a member of a thread group. Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once. They form a tree in which every thread group except the initial system thread group created at startup (called *main*) has a parent. The **ThreadGroup** class provides constructors for creating thread groups and methods that are used to manipulate the threads in the group. There is no publicly available method to add a thread to a thread group. Instead, the constructors of **Thread** can take a **ThreadGroup** instance as a parameter. If the thread group of a new thread is unspecified at construction, it is placed in the thread group of the current thread. The RTSJ does not specify the way in which thread groups may be used with **RealtimeThread** and **NoHeapRealtimeThread** objects. Therefore, as the **RealtimeThread** class extends **Thread**, the thread group semantics should be the same. The reference implementation for the RTSJ [10] does not support these semantics. It turns out that implementing

thread groups in the presence of scope-using `RealtimeThreads` would require the creation of references that would break the RTSJ rules. A primary reason for this is that `RealtimeThreads` created in scoped memory regions would have to hold a reference to the thread group instance in addition to the thread group holding a reference to its member `RealtimeThreads`. In order to obey the RTSJ rules, this would require all the `RealtimeThreads` assigned to a thread group to be created in the same scoped memory region as the `ThreadGroup` instance they are registered with. If, as in the case of `Thread` instances in regular Java, every `RealtimeThread` must belong to thread group, a `ThreadGroup` instances would have to be created for every first `RealtimeThread` instance created in a memory region. Moreover, the RTSJ scoping rules make it impossible to maintain the tree structure of thread groups and their sub-groups which requires downward references in the scopes.

In this section we show how we can implement thread groups for `RealtimeThread` objects created in scoped memory regions using scoped reference objects. We show how thread groups and their subgroups may span scopes providing full thread group functionality for RTSJ applications.

6.1 Requirements for RTSJ Thread Groups

If scoped reference objects and regular references are used together to create an arbitrary tree structure of groups with `RealtimeThreads` in any memory area, then the thread group behaviour as exhibited in Java runs into problems. Consider Figure 2 (a) which shows the thread group tree spanning scopes and with threads from different memory areas and regions registered with groups in the tree. Note that the arrows do not represent references but simply the tree structure. TG4 has two registered `RealtimeThreads` which exist in an outer and inner scope. It is clear from the figure that, as SM3 will be cleared before all other memory areas, TG4 will be left without a parent thread group. This will imply that there is no longer a single tree of thread groups in the application but a forest of trees. Moreover, as SM1 will outlive SM2, there will be a point during the application's lifetime in which the `Schedulable` in SM1 will not be registered with any valid thread group.

There are three possible semantics which we can define for this occurrence:

- Accept that RTSJ applications may have `RealtimeThreads` that may fall out of a thread group (ie that `RealtimeThread`'s thread group becomes `null`) and that subtrees of thread groups may break off from the main tree.
- Have `RealtimeThreads` become registered with their owning thread group's parent if their own thread group is removed. Similarly have thread groups become subgroups of their parent's parent if their own parent is removed.
- Require every thread group to outlive all its child thread groups and any `RealtimeThreads` registered with it.

We choose the final option as it provides the semantics most similar to thread groups in Java. We show that by making use of scoped reference objects together with regular references from scoped memory as defined by the RTSJ scoping rules, these semantics can be achieved.

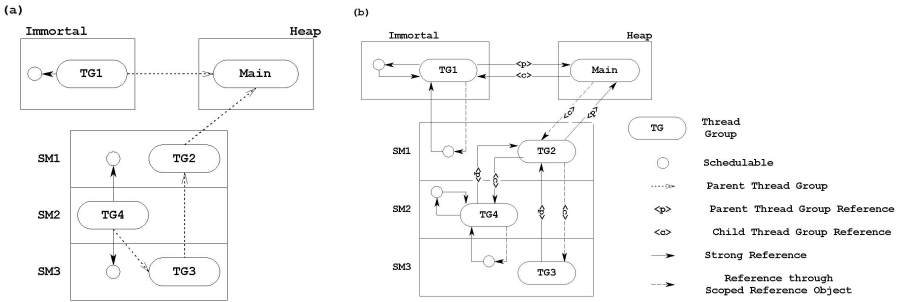


Fig. 2. Thread Groups through Scoped References

6.2 Thread Groups through Scoped References

Each thread group instance in an implementation would require (1) a reference to the parent thread group, (2) a list of references to subgroups of that group and (3) a list of references to the threads registered with that group. Each `RealtimeThread` requires (4) a single reference to the thread group it is registered with. Implementing thread groups of `RealtimeThreads` requires consideration of how each of these reference should be implemented, that is if regular strong references or scoped reference objects should be used. The requirements set out in Section 6.1 hint at which reference type should be used in each case. Since we require thread groups to outlive their parents and threads to outlive their thread group, it follows that (1) and (4) above should be regular strong references. However, since thread groups will outlive their subgroups and the `RealtimeThreads` registered with them, they will hold scoped references to these instances (3 and 4 above). Where thread groups and `RealtimeThreads` are not to be created in scoped memory, strong references can always be used.

Figure 2 (b) shows the use of the different reference types to build a thread group tree and register `RealtimeThreads` with those groups. Note that thread group TG1 can use a strong reference both for the parent and child references. On the other hand the main thread group requires a scoped reference object to reference TG2. TG1 is shown to have two `RealtimeThread` instances registered with it. The `RealtimeThread` in the same immortal memory area is referenced by a regular reference while the `RealtimeThread` in SM1 is referenced by a scoped reference object. Therefore, if a method is to be invoked on all the `RealtimeThreads` registered with TG1, the invocation on the scoped `Schedulable` instance will only succeed if the reference count of SM1 would not have reached 0 at some point after the `Schedulable` was created. Any failures in invocation would raise an exception which could be ignored.

7 Future Work and Conclusion

In this paper we have provided a detailed look at how reference objects may be used in the RTSJ. The key contribution is the development of the scoped refer-

ence object that safely allows access to objects where this would be disallowed by the RTSJ scoping rules. We have shown how scoped reference objects provide an easy solution to one of the major implementation problems of the RTSJ – the implementation of thread groups for **RealTimeThreads**.

We are looking at how the semantics of scoped reference objects can be extended to the distributed case. A key issue in implementing a distributed RTSJ using Remote Method Invocation (RMI) [11,12] is the definition of the semantics of remote references to scoped objects and how these would define the behaviour of the distributed garbage collector. We believe scoped reference objects can provide a natural extension as the proxy mechanism can be paralleled to the stub/skeleton mechanism of RMI. Finally, we are working on an implementation of scoped reference objects by modifying the RTSJ reference implementation [10].

References

1. Baker, F.: The Treadmill: Real-Time Garbage Collection without Motion Sickness. Proceedings of the Workshop on Garbage Collection in Object-Oriented Systems (OOPSLA'91) (1991)
2. Cheng, H. P., Belloch G.: A Parallel, Real-Time Garbage Collector. ACM SIGPLAN Conference on Programming Language Design and Implementation (2001) 125–136
3. Bollella, G. et. al.: The Real-Time Specification for Java. Pulished by Addison-Wesley, Reading, MA, USA. Available at: <http://www.rti.org> (2000)
4. Tofte M., Talpin J.: Region-Based Memory Management. Information and Computation **132,2** (1997) 109–176
5. Deters M., Cytron R. K.: Automated Discovery of Scoped Memory Regions for Real-Time Java. Proceedings of the Third International Symposium on Memory Management (2003) 25–35
6. Beebe W., Rinard M.: An Implementation of Scoped Memory for Real-Time Java. First International Workshop on Embedded Software (EMSOFT) (2001)
7. The Reference Object API for Sun's JDK 1.3 – Available at: <http://java.sun.com/j2se/1.3/docs/guide/refobs/>
8. Herlihy M.: Wait-Free Synchronization. ACM Transactions on Programming Languages and Systems **11,1** (1991) 123–149
9. Higuera M., Issarny V., Banatre M., Cabillic G., Lesot J., Parain F.: Region-based Memory Management for Real-time Java. Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC) (2001)
10. TimeSys' Real-Time Specification for Java Reference Implementation. Available at: <http://www.timesys.com/rtj>
11. Wellings A. J., Clark R., Jensen D., Wells D.: A Framework for Integrating the Real-Time Specification for Java and Java's Remote Method Invocation. Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC) (2002) 13–22
12. Borg A., Wellings A.: A Real-Time RMI Framework for the RTSJ. Proceedings of the 15th Euromicro Conference on Real-time Systems (2003)

Deriving Java Virtual Machine Timing Models for Portable Worst-Case Execution Time Analysis

Erik Yu-Shing Hu, Andy J. Wellings, and Guillem Bernat

Real-Time Systems Research Group
Department of Computer Science
University of York, York, YO105DD, UK
{erik,andy,bernat}@cs.york.ac.uk

Abstract. Performing worst-case execution time (WCET) analysis on the highly portable real-time Java architectures without resulting in the under utilisation of the overall system has several challenges. Current WCET approaches are tied to either a particular language or target architecture. It should also be stressed that most WCET analysis approaches are usually only considered in relation to procedural programming languages. In this paper, we propose a comprehensive portable WCET analysis approach, and demonstrate how Java virtual machine timing models can be derived effectively on real-time and embedded Java-based systems.

Keywords: Real-Time Java, Worst-Case Execution Time (WCET) Analysis, Portable WCET

1 Introduction

To be able to support a predictable and expressive real-time Java environment, two major international efforts have attempted to provide real-time extensions to Java: the Real-Time Specification for Java (RTSJ) [4] and the Real-Time Core extensions to Java [6]. These specifications have addressed the issues related to using Java in a real-time context, including scheduling support, memory management issues, interaction between non-real-time Java and real-time Java programs, and device management, among others.

On the whole, timing analysis is crucial in real-time systems to guarantee that all hard real-time threads will meet their deadlines in line with the design. In order to ensure this, appropriate scheduling algorithms and schedulability analysis are required. Typically, most scheduling algorithms assume that the *Worst-Case Execution Time* (WCET) estimation of each thread has to be known prior to conducting the schedulability analysis. Therefore, estimating WCET bounds of real-time threads is of vital importance. Unfortunately, in neither of the real-time Java specifications [4,6], is there a satisfactory solution to how WCET estimations can be carried out on the Java architecture with portability in mind.

For the most part, current WCET approaches [3,5,14,16,17] are tied to either a particular language or target architecture. In addition, these analysis approaches are only considered in relation to procedural programming languages. Performing WCET analysis on the Java architecture must take into account not only additional dynamic features, such as dynamic dispatching and memory management, but also the platform independent issue. Therefore, most WCET analysis approaches are not appropriate for the Java architecture, since Java programs are “write once, run every where” or perhaps more appropriate for real-time “write once carefully, run everywhere conditionally” [4].

To our knowledge, only the portable WCET analysis proposed by Bernat et al. [2,1] has taken account of platform independent features for the Java architecture. Portable WCET analysis uses a three-step approach: high-level analysis (i.e. analysing the Java programs), low-level analysis (i.e. producing a *Virtual Machine Time Mode*¹ (VMTM) for a target platform), and conducting a combination of the high-level analysis with the low-level analysis to compute the actual WCET bound of the analysed code sections. However, the portable WCET analysis approach has only tended to focus on supporting portability, rather than addressing the dynamic features of Java. It also should be noted that portable WCET analysis highly depends on the VMTM of a target platform, and there is also no appropriate solution to show how a VMTM for a particular platform can be built efficiently. Therefore, from a practical standpoint, bringing this approach into engineering practice still has a number of issues to be addressed.

For the above reasons, we introduce a comprehensive portable WCET analysis that takes into account the dynamic dispatching issues [7,10,11] and presents how VMTMs can be built for various target platforms. The major contributions of this paper are:

- introducing two measurement approaches that demonstrate how to extract Java VMTMs for portable WCET analysis,
- discussing how VMTM can be derived by a profiling-based approach, and
- presenting how to build a portable benchmark model to extract VMTMs from various target platforms.

The rest of the paper is organised as follows. Section 2 gives a summary of the related work, while Section 3 presents an overview of the comprehensive portable WCET analysis. Section 4 discusses how Java VMTMs can be extracted from various platforms with two different approaches. Following this, Section 5 gives a simple example to evaluate our approaches. Finally, conclusions and future work are presented in Section 6.

2 Related Work

In general, there are two principal ways for obtaining the WCET of a program: static analysis and dynamic analysis (a.k.a. measurement approach). Most systems in industry have relied on ad-hoc measurements of execution times when

¹ VMTM is a timing model for the target virtual machine including a list of the worst-case execution time of native methods and Java bytecode instructions.

designing real-time systems [15]. Arguably, measuring an execution time could be an unsafe practice, since one cannot know whether the worst case has been captured in the measurements.

In contrast, a static analysis could give relatively safer results for the WCET analysis [18]. A number of research approaches [5,14,16,19] have demonstrated how to estimate WCET at high-level and low-level analyses. Unfortunately, the above WCET approaches are tied to either a particular language or target architecture. In addition, these analysis approaches are only considered in relation to procedural programming languages.

As processors have tended to be more complex recently, some research approaches [3,15,17] have integrated measurement techniques with static analysis to address modern complicated processor issues. However, these approaches have attempted to estimate WCET bounds from applications to the target platform at once. As a result, these techniques cannot take advantage of the platform independent feature supported in Java.

Notable exceptions include the portable WCET analysis proposed by Bernat et al. [2,1]. This approach has taken into account platform independent features for the Java architecture. However, they have only tended to focus on supporting portability, rather than taking account of the issues connected with the use of dynamic dispatching features. Nor is there an appropriate solution to show how a VMTM for a particular platform can be built efficiently.

Our approach extends the portable WCET analysis approach to take into account dynamic dispatching issues and provide a portable model to build VMTM effectively on real-time and embedded systems. Arguably, there is some additional pessimism in performing the WCET process in this particular way, which counteracts for the added benefits that portability brings [2]. It can be observed that this pessimism can be compensated with the use of gain time reclaiming mechanisms [10,11] integrated in our approach.

3 Overview of the Framework

Following the philosophy of portable WCET analysis [2,1], our framework (Fig. 1), therefore, also uses the three-step approach to be able to offer a comprehensive WCET analysis bearing portability and dynamic dispatching issues in mind.

Note that this framework is part of our on-going work, called XRTJ² [9], which extends the current Real-Time Java architecture [4] proposed by the Real-Time Java Expert Group. The XRTJ environment is targeted at cluster-based distributed high-integrity real-time Java systems, such as consumer electronics and embedded devices, industrial automation, space shuttles, nuclear power plants and medical instruments. In the XRTJ environment, to facilitate the various static analysis approaches and provide information that cannot be expressed in either Java source programs or Java bytecode, an extensible and

² XRTJ: an Extensible Distributed High-Integrity Real-Time Java Environment

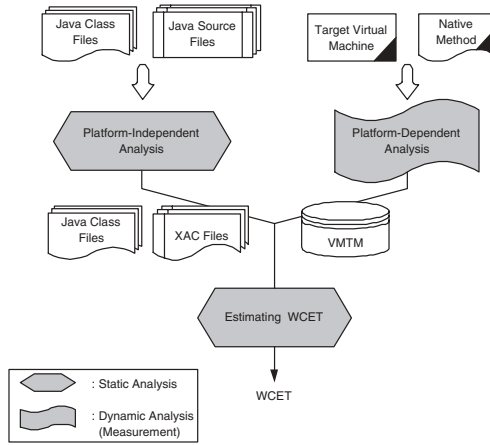


Fig. 1. The comprehensive portable WCET framework

portable annotation class format called *Extensible Annotations Class* (XAC) file is proposed [8]. To generate XAC files, an annotation-aware compiler, named *XRTJ-Compiler* [9], which can derive additional information from either manual annotations or source programs, or both, is also introduced. Furthermore, a static analyser, called *XRTJ-Analyser* [9], is introduced in order to support various static analyses, including program safety analysis and timing analysis.

The first step of the framework is the *platform-independent* analysis. At this stage, the technique analyses annotated Java programs or Java class files to produce portable WCET information. Manual annotations in our approach are introduced with the characters ‘//@’ for single line and ‘/*@ ... @*/’ for multiple lines [8]. Note that these formats are assumed to be comments in Java. Taking advantage of the knowledge accumulated with the compiler, portable WCET information can be extracted from either source programs or Java byte-code statically. Here, dynamic dispatching methods can also be analysed with our previous approaches [7,10,11]. Portable WCET information is computed in the form of so-called *Worst-Case Execution Frequency* (WCEF) vectors by the XRTJ-compiler. WCEF vectors [1] represent execution-frequency information about basic blocks³ and more complex code structures that have been collapsed during the first part of the analysis. Then portable WCET information can be stored into the XAC files [8]. Note that the static analysis is used in this stage.

In parallel, analysis of the target platform is performed, so-called *platform-dependent* analysis. This takes the form of the definition of a timing model of the virtual machine. This stage performs platform-dependent analysis (i.e. in the context of specific hardware and VM) of the implementation of Java

³ A basic block is a continuous section of code in the sense that control flow only goes in at the first instruction and leaves through the last one.

bytecode instructions. During this stage, information about the potential effects of pipelines [1] and caches⁴ may be captured.

Although the platform-independent analysis can be carried out by a static analysis approach, the use of a static analysis technique to perform the platform-dependent analysis has a number of challenges. It should be noted that when deriving VMTM it is necessary to take into account the implementation aspects of not only the Java virtual machine, but also the operating system. In addition, the analysis should also bear in mind the hardware architecture for various embedded systems. In order to accommodate a diverse set of implementations on the underlying platforms and virtual machines for embedded systems the measurement-based analysis technique is used in our approach. This will be explored further in Section 4.

The final stage is the estimating of the WCET bounds of each thread. In the XRTJ environment, a WCET analysis tool in the XRTJ-Analyser performs the combination of the results of platform-independent analysis with the target VMTM to compute the actual WCET bound of the analysed code sections.

4 Deriving Java Virtual Machine Models

Deriving the VMTM of a target platform is crucial in the portable WCET analysis since the results of the analysis are highly dependent on the outcome of the VMTM. Arguably, in the real-time and embedded field, analysing a virtual machine to produce the VMTM of the target platform needs to be efficient and rapid since the development life-cycles of the software built for embedded systems are short and the applications are demanded to be reusable and compatible among various architectures. Therefore, how to efficiently derive VMTMs for different platforms is the key issue for the portable WCET analysis approach. In this paper, we propose two measurement approaches: *profiling-based analysis* and *benchmark-based analysis*; which demonstrate how the VMTM can be extracted from a target platform.

Note that there are several possible ways in which the execution time can be measured, such as using *clock cycle counters* and using *timers*. In our approach, we use the *rdtsc* instruction, which has high resolution and very low overhead at run-time, provided in x86 architecture [12] to extract the time-stamp counter of the processor. Although we only show the use of software approach on the x86 architecture under the Linux platform here, our approach can also be applied to other CPU architectures and operating systems if they support instructions or libraries that can extract the time-stamp counter of the processor. For example, *getrtime()* library routine can be used on the SPARC V9 architecture under Solaris 8 operating systems, and hardware data acquisition interfaces⁵ can be used under Windows, Linux and Solaris operating systems.

⁴ The analysis of the cache effects is our future work and beyond the scope of this paper.

⁵ <http://www.ultraviewcorp.com/>

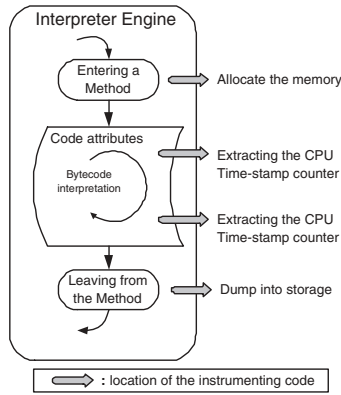


Fig. 2. Instrumenting profiling code into an interpreter engine

4.1 Profiling-Based Analysis

Observing the behaviour of a system to analyse the specific aspects of applications executing on the system is not novel. An automatic tracing analysis [20] has been proposed to extract temporal properties of applications and operation systems. The approach shows that the empirical analysis can reduce the over-estimation of real-time applications. Accordingly, a profiling-based analysis technique can be applied to deriving a VMTM for a particular platform by instrumenting additional code into the virtual machine.

Even though the idea is relatively straightforward to derive a VMTM, a number of issues need to be addressed to ensure the reliability of the derived VMTM. For example:

- where to insert the instrumenting code,
- how to minimise the side effects of the instrumenting code at run-time, and
- how to avoid the out-of-order execution during the measurement of the specific code section.

Similar to the automatic tracing analysis approach [20], profiling the execution time of each bytecode can be divided into two steps. One is extracting run-time information and the other is analysing it. The former step involves: exploring the context of the virtual machine where temporal information can be derived; and the instrumenting code to extract the time-stamp counter of the processor with very low runtime overhead. For instance, the instrumenting code has to accumulate the instruction mnemonics and the time-stamp counter every time the interpreter fetches a bytecode. The latter step analyses these data and builds up a VMTM for the target platform.

To be able to trace the run-time information, instrumenting code needs to be provided into the Java virtual machine. The instrumenting mainly depends on the specific implementation of the JVM. However, on the whole, Java virtual machines conduct the interpretation of Java bytecode by a method-based

approach. Therefore, to reduce the memory and run-time overhead needed for collecting the run-time information, the implementation of the profiling-based analysis can refer to a suggested implementation, given in Fig. 2. Note that the major aim of the collecting run-time data by method in the interpreter engine is only to reduce the memory and run-time overhead of the instrumenting code, rather than analysing the applications.

As shown in the figure, a small amount of memory, which can be allocated when invoking a method, is necessary to store the collected information during run-time. These accumulated data can be dumped into storages when returning from the method (i.e. finishing the interpretation of the method). Dumping accumulated data at this point can reduce the noise or side effects of the instrumenting code on the measurement results. Here, these data can be analysed with the requirements of the target platform and the VMTM can be built with these analytical data. To avoid out-of-order execution during profiling, a serializing instruction (*cpuid*) can be invoked before extracting the time-stamp counter of the processor.

The experimental implementation of this approach has been carried out on the reference implementations of RTSJ provided by TimeSys [21]. Basically, the instrumenting code, including the serializing and time-stamp counter instructions, is added into the interpreter engine. Before starting the interpretation of a method, a buffer to store run-time information is prepared. Then, the execution time of each bytecode can be measured starting from the opcode fetched to before fetching the next opcode. The run-time information captured by the interpreter is classified by the opcode mnemonics. When leaving the method, the captured run-time information can be conducted with statistical analysis to produce the VMTM. The evaluation of this approach is discussed in Section 5.

4.2 Benchmark-Based Analysis

It should be noted that the analysis of the portable WCET analysis approach highly depends on the VMTM of a target platform, and the technique provided in the previous section needs enormous effort to be carried out, including modifications to the execution engine of the target Java virtual machine to derive the execution time of each bytecode. In order to conduct this, it is clear that the source of the virtual machine is necessary. Although deriving the execution time of a single bytecode can be achieved by the previous mechanism, deriving the execution of specific sets of bytecodes is unlikely to be accomplished. Furthermore, the implementations of the previous approach cannot be reused for building the VMTM of a new virtual machine. This means that creating a VMTM for a new virtual machine needs to be started from scratch. Therefore, to be able to apply portable WCET analysis to real-time and embedded systems effectively, two major issues need to be addressed:

- how the instrumenting code can be reused effectively on various platforms without modifying it, and
- how the execution time of a specific set of bytecodes can be measured.

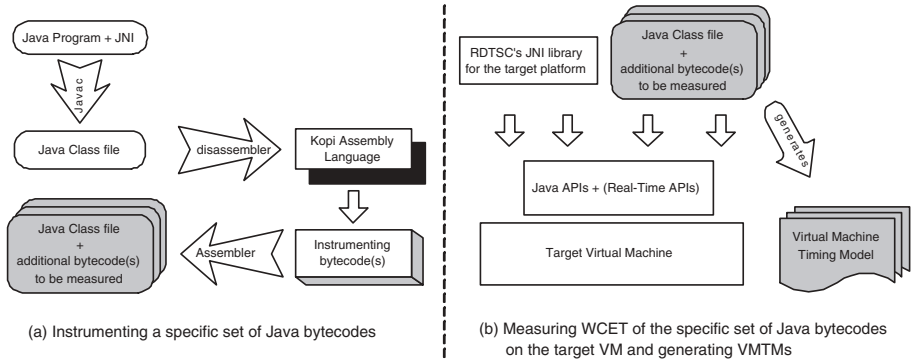


Fig. 3. Instrumenting and Measuring of the benchmark-based approach

To address these issues, the *benchmark-based analysis* approach is introduced. The aim of this approach is to provide a Java-based benchmark⁶ that may produce a VMTM automatically after executing it on the target virtual machine. The principle behind this mechanism is to inject individual or a set of specific bytecodes into the instrumenting code developed in a native method that may access the time-stamp counter of the processor in a Java program. Therefore, the native method using Java native interface (JNI) features in the benchmark can be ported easily to different platforms without modifying the benchmark. However, some issues need to be addressed to achieve these goals:

- where and how specific bytecodes can be inserted into the Java program to measure the execution time of the specific bytecodes, and
- how to maintain the integrity of the Java stack after the injection of additional bytecodes.

To prove the feasibility of this approach and reduce the time needed to develop the whole mechanism, a number of tools have been investigated. Taking advantage of the time-stamp counter instruction (*rdtsc*) [12] supported in x86 architecture, the bytecodes *disassembler* and *assembler* tools provided in the *Kopi Compiler Suite* [13], and the Java native interface feature, the benchmark-based analysis approach can be carried out. The procedure of how the benchmark can be established is given below.

As shown in Fig. 3, a Java program with a native method that can access the time-stamp counter can be translated into Java bytecode by a traditional compiler. Then, the class file can be translated into Kopi assembly language to be able to insert a specific set of Java bytecodes easily under text mode. Here, one should note that the integrity of the Java stack of JVM needs to be borne in mind when inserting additional bytecodes. For instance, after executing the *iload* instruction, the virtual machine will load an integer onto the Java stack.

⁶ The term *benchmark* means a collection of Java programs that are instrumented with particular bytecodes to be measured.

Table 1. Measurements of the WCET of the instrumenting code

Experiment	99.95%	99.90%	Average
1.	334	321	320
2.	326	321	321

Table 2. Measurements of the WCET of *iload* with the instrumenting cost

Experiment	99.95%	99.90%	Average
1.	366	353	353
2.	363	353	353

Therefore, we need to add complementary bytecodes to remove the integer from the stack in order to maintain the data integrity of the Java stack for the whole program. Some bytecodes may also need to be provided with values or references before executing them, such as *iadd* and *iaload*. As a result, to ensure the data integrity of the Java stack, corresponding complementary bytecodes needed to be added at the *pre*- or *post*- locations of the measurement bytecodes. After injecting the specific bytecodes, the file saved in the Kopi assembly language format can be translated into standard Java class files. As presented in Fig. 3, these individual instrumenting Java programs can be combined together into a comprehensive benchmark that can generate VMTM automatically. Then, the individual Java program or the benchmark is ready to be used for measuring the execution time of the specific set of bytecodes on any target platform.

One should note that the major purpose of the benchmark is to produce VMTM that contains a collections of the WCET bounds of WCEF vectors and method calls. Therefore, it is possible that building VMTM to be used in systems in the small can provide a compact benchmark that comprises those WCEF vectors that will only be used on such systems. The benchmark can be executed on any particular target platform with a native method that can access the time-stamp counter of the target platform. This approach can be used to generate the execution time of a specific set of common sequence bytecodes since it allows to insert any combination of bytecodes with this mechanism. It can be observed that generating instrumented Java programs can be automatically conducted by a simple program implementing the above procedure.

An experiment has been carried out on the RTSJ-RI and the preliminary result of the analysis of *iload* bytecode is given below. A Java program developed with a native method that can access the time-stamp counter of the processor is developed to measure the cost of the instrumenting code. The experimental results of the cost of the instrumenting code is given in Table 1. Then, *iload* bytecode instructions is added into the instrumenting code and the corresponding complementary bytecode (i.e. *istore*) is inserted at the post instrumenting code. The measurement of the *iload* has been carried out in a 50000-times loop. The experiment has been conducted several times and two of them are given in Table 2. There is a graph illustrating that the distributions of the measurement of the *iload* instruction is given Figure 4. The *iload:100%* shows the machine cycles of all the measurements in the loop, whereas the *iload:99.95%* presents 99.95 percentage of the measurements of the loop where we assumed that very high execution time of the measurements are influenced by the operation systems and interrupts.

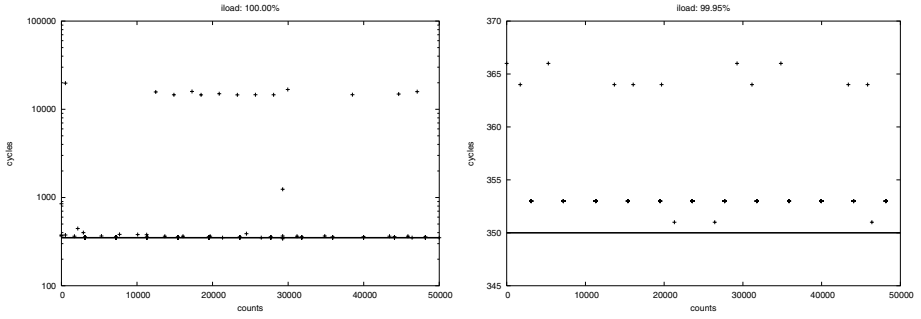


Fig. 4. Measurements of the *iload* bytecode with the benchmark-based analysis

5 Evaluation

The evaluation of our analysis is illustrated with an example code of the Bubble Sort algorithm presented in Fig. 5. Figure 6 shows the individual basic block of the algorithm with offset numbers. The maximum number of iterations of the outer and inner loops can be assumed as $10 - 1$ and $10(10 - 1)/2$ respectively when the *size* is equal to 10.

The WCEF vectors of the bubble sort algorithm, generated by our prototype compiler during compilation, is given in Fig. 7 in text mode. In this example, only 14 different Java bytecodes are generated by the Java compiler. When deriving VMTM it is necessary to minimise the run-time overhead and influences of background process running in the operating systems including background tasks and interrupters. We addressed these issues by running the test-bed under single user mode on Linux. In addition, other background processes are killed manually to reduce the influences as much as possible. The measurements of

```
import javax.realtime.*;
public class rth extends RealtimeThread {
    static int Data[];
    public void run() {
        // ...
        bb_Sort(Data);
        waitNextPeriod();
    }
    public static void bb_Sort(int a[]) {
        int i, j, t;
        int size = 10;
        for(i=size-1; i>0; i--) {
            for(j=1; j<=i; j++) {
                if(a[j-1] > a[j]) {
                    t = a[j-1];
                    a[j-1] = a[j];
                    a[j] = t;
                }
            }
        }
    }
}
```

Fig. 5. The Bubble Sort Algorithm in Java

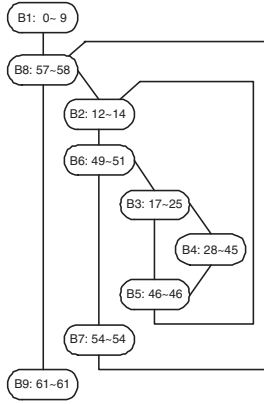


Fig. 6. Individual basic blocks with their offset numbers

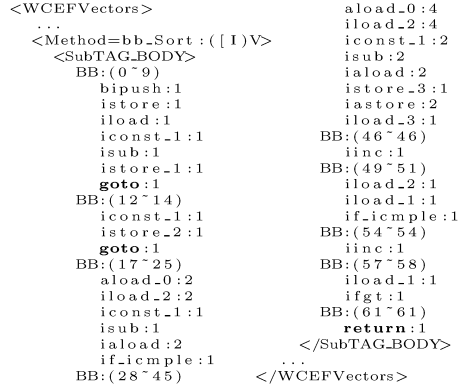


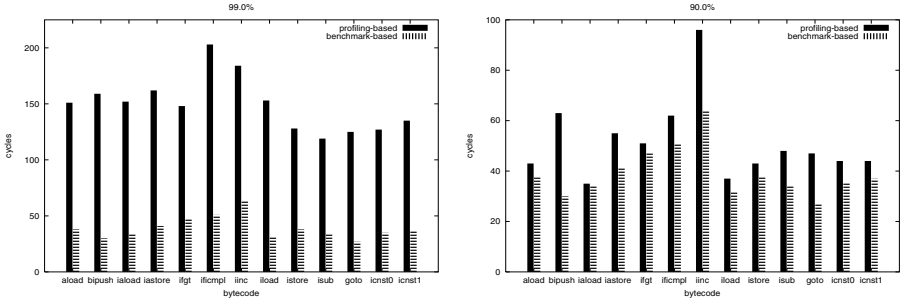
Fig. 7. WCEF Vectors of the bubble sort algorithm in text mode

the execution time are represented with machine cycle unit in the rest of this example.

A summary of the VMTM for the Bubble Sort example is shown in Table 3. This table shows the different statistical analysis results of the VMTM carried out with benchmark-based analysis. Each bytecode is measured by 50000 times continuously. As shown in Fig. 8, although the VMTM derived with the benchmark-based approach shows rather constant outcomes, the VMTM produced with the profiling-based approach presents relatively pessimistic results if the 99% of the measurements have been taken into account as the WCET bounds. This can be reasoned that the ad-hoc measurements of the profiling-based analysis can produce pessimism because it derives the execution time of each bytecode from various methods invoked on the VM and most methods are invoked during the initialisation phase of the VM. As a result, some measurements could be the worst-case response time of the bytecode instead of the WCET bounds.

Table 3. A VMTM derived with the benchmark-based analysis

Bytecode	99.95%	99.90%	Average
aload	48	38	36
bipush	40	30	30
iaload	46	34	36
iastore	55	41	31
ifgt	67	47	27
if_icmple	71	51	52
iinc	93	64	62
iload	37	32	24
istore	50	38	36
isub	45	34	35
goto	36	27	25
iconst0	38	35	31
iconst1	37	37	32

**Fig. 8.** Comparing the profiling-based and benchmark-based analyses**Table 4.** Comparing the final WCET bounds

Approach	99.9%	90.0%
End-to-end measurement	40378	39865
Benchmark-based	42125	39908
Profiling-based	164689	51235

However, it can be observed that the 90% percent of the measurements of the profiling-based analysis are very close to the results derived with the benchmark-based analysis. The experiment also shows that the profiling-based analysis has some difficulties to control which particular bytecodes to be measured and the number of the measurements of the bytecodes. Therefore, in order to obtain the reliable measurements with the profiling approach, it also needs to be provided with a large amount of the particular bytecodes needed to be measured.

Using Table 3, three different WCET bounds (i.e. 99.95%, 99.90%, and average) can be estimated. The WCEF of the bubble sort algorithm is obtained as follows:

$$\begin{aligned} \text{WCET}(\text{bb\_Sort}()) \\ = B1 + 10*B8 + 9(B2+B7) + 46*B6 + 45(B3+B4+B5) + B9 \end{aligned}$$

The final WCET bounds of the algorithm with different approaches (i.e. end-to-end measurement, benchmark-based analysis and profiling-based analysis) have been conducted. The estimations taking account of the 99% and 90% of the measurements in Table 4. Note that the method of estimating the pipeline effects is beyond the scope this paper and the technique proposed in [1] can be integrated into our approach easily with benchmark-based analysis.

6 Conclusion and Future Work

Since the aim of portable code is to support hardware interchangeability, the WCET analysis for such portable applications needs to bear portability in mind. The comprehensive portable WCET has been proposed with a three stage approach to analyse the highly portable and reusable Java applications for real-time

and embedded systems. In this paper, we have mainly discussed how to derive various VMTMs to facilitate the use of portable WCET [2] in real-time and embedded Java-based applications.

Two approaches (i.e. profiling-based and benchmark-based) have been proposed to derive VMTMs. The major advantage of the profiling-based approach can be extended to integrate with other tracing or profiling techniques, such as POSIX-trace [20], whereas the disadvantages of the approach are that it needs the source code and knowledge of the target VM and it takes time to instrument the additional code into the VM. In contrast, the benchmark-based analysis is highly portable and only needs to provide a native method to access the timestamp counter of the target processor. However, the benchmark-based analysis is less convenient to integrate with other profiling techniques. Therefore, these techniques can be applied to various applications that depend on the requirements of the systems.

Based on the experimental results, the outcomes of the benchmark-based analysis approach (Figure 8) encourage us to carry on the future work on the use of portable WCET analysis in real-time and embedded Java-based systems, whereas the results of the profiling-based analysis approach reminds that taking account of other run-time issues, such as cache effects and branch prediction issues, can achieve relatively safer and tighter WCET estimations.

There are still a number of issues that need to be addressed in our approach, such as taking into account the timing properties of the RTSJ, cache effects on WCET estimations and extending for just-in-time compiler techniques.

References

1. I. Bate, G. Bernat, G. Murphy, and P. Puschner. Low-Level Analysis of a Portable Java Byte Code WCET Analysis Framework. *Proceedings of the 6th IEEE Real-Time Computing Systems and Applications RTCSA-2000*, pages 39–48, December 2000.
2. G. Bernat, A. Burns, and A. Wellings. Portable Worst-Case Execution Time Analysis Using Java Byte Code. *Proceedings of the 6th Euromicro Conference on Real-Time Systems*, pages 81–88, June 2000.
3. G. Bernat, A. Colin, and S. Petters. WCET Analysis of Probabilistic Hard Real-Time Systems. *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, December 2002.
4. G. Bollella, J. Gosling, B. M. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. *Real-Time Specification for Java*. Addison Wesley, 2000.
5. R. Chapman, A. Burns, and A. Wellings. Integrated Program Proof and Worst-Case Timing Analysis of SPARK Ada. *Proceedings of the Workshop on Language, Compiler, and Tool Support for Real-Time Systems*, June 1994.
6. J. Consortium. Real-Time Core Extensions for Java platform. *International J Consortium Specification*, Revision 1.0.14, September 2000. <http://www.j-consortium.org/rtjwg/>.
7. E. Y.-S. Hu, G. Bernat, and A. J. Wellings. Addressing Dynamic Dispatching Issues in WCET Analysis for Object-Oriented Hard Real-Time Systems. *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC-2002*, pages 109–116, April 2002.

8. E. Y.-S. Hu, G. Bernat, and A. J. Wellings. A Static Timing Analysis Environment Using Java Architecture for Safety Critical Real-Time Systems. *Proceedings of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems WORDS-2002*, pages 77–84, January 2002.
9. E. Y.-S. Hu, J. Kwon, and A. J. Wellings. XRTJ: An Extensible Distributed High-Integrity Real-Time Java Environment. *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications RTCSA-2003*, pages 371–391, February 2003.
10. E. Y.-S. Hu, A. J. Wellings, and G. Bernat. A Novel Gain Time Reclaiming Framework Integrating WCET Analysis for Object-Oriented Real-Time Systems. *Proceedings of the 2nd International Workshop on Worst-Case Execution Time Analysis WCET-2002*, June 2002.
11. E. Y.-S. Hu, A. J. Wellings, and G. Bernat. Gain Time Reclaiming in High Performance Real-Time Java Systems. *Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC-2003*, pages 249–256, May 2003.
12. Intel's Applications Notes. Using the RDTSC Instruction for Performance Monitoring. Intel. <http://cedar.intel.com/software/idap/media/pdf/rdtscpm1.pdf>.
13. Kopi. The Kopi Project. DMS Decision Management Systems Gmb. [Hhttp://www.dms.at/kopi/](http://www.dms.at/kopi/).
14. S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Kim. An accurate worst case timing analysis for RISC processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, July 1995.
15. M. Lindgren. Measurement and Simulation Based Techniques for Real-Time Systems Analysis. Dissertation, Uppsala University, Sweden, 2000.
16. F. Mueller. *Static Cache Simulation and its Applications*. Ph.d thesis, Department of Computer Science, Florida State University, July 1994.
17. S. Petters and G. Farber. Making Worst Case Execution Time Analysis for Hard Real-Time Tasks. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Application RTCSA-1999*, December 1999.
18. P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.
19. A. Shaw. Reasoning about Time in Higher-Level Language Software. *IEEE Transactions on Software Engineering*, 15(7):875–889, July 1989.
20. A. Terrasa and G. Bernat. Extracting Temporal Properties from Real-Time Systems by Automatic Tracing Analysis. *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications RTCSA-2003*, pages 483–502, February 2003.
21. TimeSys. Real-Time Java. TimeSys. <http://www.timesys.com/prodserv/java/>.

Efficient Bytecode Verification Using Immediate Postdominators in Control Flow Graphs^{*}

Extended Abstract

Cinzia Bernardeschi, Nicoletta De Francesco, and Luca Martini

Dipartimento di Ingegneria dell'Informazione, Università di Pisa,
Via Diotisalvi 2, 56100 Pisa, Italy
{cinzia,nico,luca.martini}@iet.unipi.it

1 Introduction and Motivations

Java Virtual Machine (JVM) code (also called bytecode) [11] can be downloaded by untrusted sources and executed directly. A component of the Java security model is the Java bytecode Verifier, which makes sure that the bytecode is safe by static analysis at loading time and prior to execution. The bytecode Verifier checks type correctness, stack overflow or underflow, code containment, registers and objects initialization, access control. The first bytecode verification algorithm is due to Gosling and Yellin. Most JVM instructions are typed and the algorithm is based on an abstract interpreter for the instructions that operates over types instead of values. Almost all existing Verifiers implement this algorithm [4,11,12].

Verification is a complex process and it is expensive both in space and time. When the Java technology runs on smart cards (Java cards) or other devices with limited memory, the memory requirements for the verification can often make infeasible the verification of the bytecode directly on such systems. In this case, the bytecode verification is generally executed before installing the bytecode on these embedded systems. Considerable research efforts have been expended to propose new techniques to solve the problem of bytecode verification on Java cards.

This paper exploits immediate postdominators in control flow graphs as a means of reducing the memory required by the bytecode verification process. The proposed approach contributes to solve the problem of bytecode verification for small memory-sized systems, like Java cards. The problem of the complexity in space of the Verifier is due to the fork and join points in the code because of branching instructions. The Verifier must propagate the inferred types to all possible successors of the forking instructions. Moreover, as the target of one or several branches can be reached along several paths, the Verifier has to make sure that the types of the stack and the registers agree at the join point. This

^{*} This work was partly supported by the Italian “Fondazione Cassa di Risparmio di Pisa” within the project “Sviluppo di un verificatore per la sicurezza dei dati in smart card multi-applicazione” (2002-04)

can be obtained by using a data structure, called *dictionary*, associating a stack and register type to each program point i . This dictionary entry represents the state in which the instruction i is executed, also called the *before state* of i . The dictionary entry for a branch target always contains the least upper bound of stack and register types inferred on all branches of control that lead to that instruction. Whenever an instruction is abstractly executed, the state after its execution is calculated and it is merged with the dictionary entry of each of its successors. If the merge result is different from the original dictionary entry, the instruction is inserted in a *working list* that contains all the instructions still to be verified. The Sun's bytecode Verifier maintains a dictionary entry to each program point that is the target of a conditional or unconditional branch.

The immediate postdominator of an instruction (*ipd*, for short) in the control flow graph of the program represents the first instruction common to all the execution paths starting at such instruction. We propose a verification algorithm that delays the execution of the instruction corresponding to the immediate postdominator of a *conditional branch* until all the different execution paths between the conditional branch and its immediate postdominator are verified. At this point of the verification process, the *dictionary* entries needed to verify that part of the bytecode can be reused. Immediate postdominators allow to reduce the number of program points that need an entry in the dictionary by reusing entries linked to program points already verified. The paper shows that the maximum size of the dictionary used by the proposed algorithm is related to the maximum number of different targets or immediate postdominators of conditional branches that can affect the execution of an instruction of the bytecode. No entry is needed for the target of unconditional branching instructions.

The algorithm performs a verification similar to the polyvariant verification used to check the subroutines in the Javacard off card Verifier [7]. While in that approach a *contour* is used to trace the calls to the subroutines, we use a *context* to trace the *if* instructions in the bytecode that affect the instruction we are verifying, i.e. the open *if* instruction. The *context* maintains the *if* instruction executed that have not yet reached their *ipd* and makes possible an abstract interpretation that is more close to the real control flow of the program. The *context* is a stack, since along an execution path immediate postdominators are reached in reverse order with respect to the corresponding conditional branching instructions. Whenever we encounter, during the abstract interpretation, an *if* instruction, we push onto the context a pair that keeps in memory informations on which *if* has been executed and the number of alternatives that has been verified. We push off from this stack whenever we are going to execute an instruction that is an *ipd* of an *if* for which all branches have been verified.

The remainder of this paper is organized as follows: Section 2 recalls the notions of control flow graph and immediate postdominator. Sections 3 describes the standard bytecode verification. Section 4 illustrates a verification algorithm that exploits the *ipd* notion. Moreover, an estimation of the memory needed by the proposed algorithm is shown. Finally, Section 5 reports related work and conclusions.

$\alpha\text{const } c$	Push constant c with type α onto the stack
$\alpha\text{load } x$	Push the value of register x , of type α , onto the stack
$\alpha\text{store } x$	Pop off the stack a value of type α and store it into local register x
$\text{ifeq } L$	Pop an integer value off the stack and, if the value is equal to 0, branch to L
$\text{goto } L$	Jump to L
αreturn	Pop a value of type α on top of the stack and return it

Fig. 1. Instruction set

2 Control Flow Graph and Postdominators

The JVM [11] is a stack machine manipulating an operand stack, a set of local registers for each method and an heap containing object instances. In Figure 1 we show a restricted set of JVM instructions used to illustrate our approach. We assume α denotes the basic types **int** and **address**. Most instructions pop their arguments off the stack and push back their results on the stack. Local registers can be accessed via **load** and **store** instructions that push the value of a register onto the stack or store the top of the stack into a register. Control is handled by *unconditional* branch (**goto**) and *conditional* branch (**if**).

The bytecode of a method is a sequence β of instructions. Given a bytecode β , let us assign a label to each instruction, so that each instruction $\beta[i]$ has the form $i : \text{instr}$, where i is the label and instr is the instruction. We use $\#\beta$ to denote the length of β , i.e. the number of elements of β , and $\forall i \in \{0, \#\beta - 1\}$ we assume that $\beta[i]$ denotes the i -th element of β . Let $\mathcal{L}(\beta)$ be the set of labels of the instructions in bytecode β : $\mathcal{L}(\beta) = \{0, 1, \dots, \#\beta - 1\}$.

We now recall the notion of control flow graph of a program, containing the control dependencies among the instructions of the program [1].

Definition 1 (control flow graph). *Given a program β , the control flow graph of the program is the directed graph (V, E) , where $V = \{\mathcal{L}(\beta) \cup \{\text{END}\}\}$ is the set of nodes; and $E \subseteq V \times V$ contains the edge (i, j) if and only if*

- $j = i + 1 \wedge (\beta[i] \neq \text{goto } L)$
 $j = L \wedge ((\beta[i] = \text{goto } L) \vee (\beta[i] = \text{ifeq } L))$
the instruction with label j can be immediately executed after that with label i ;
- $\beta[i] = \alpha\text{return}$ and $j = \text{END}$
*there is an edge from **return** to the node **END***

*The node **END** is the final node of the graph and does not correspond to any instruction.*

We consider programs with unstructured control flow satisfying the restriction that a **return** is reachable from every instruction. Thus the final node of the control flow graph is reachable from any other node. We use the control flow graph and the notion of immediate postdominator to derive the set of conditional branching instructions that can influence the execution of a bytecode instruction i .

Definition 2 (postdominator). *Let i and j be nodes of the control flow graph of a program. We say that node j postdominates i , denoted by $j \text{ pd } i$, if $j \neq i$ and j is on every path from i to the final node. We say that node j immediate postdominates i , denoted by $j = \text{ipd}(i)$, if $j \text{ pd } i$ and there is no node r such that $j \text{ pd } r \text{ pd } i$.*

If instruction at label i is a conditional instruction, $\text{ipd}(i)$ is the first instruction belonging to all the branches originated by i . Let us call $\text{scope}(i)$ the set of instructions that are executed conditionally, according to the the value of the expression tested by a conditional instruction at label i . All the instructions in a path from i to $\text{ipd}(i)$ (exclusive) belong to the scope of the conditional instruction:

$$\text{scope}(i) = \{k \mid k \text{ belongs to a path from } i \text{ to } \text{ipd}(i), k \neq \text{ipd}(i), k \neq i\}$$

Given an instruction k , we denote by $\text{dep}(k)$ the set of conditional instructions having k in the scope

$$\text{dep}(k) = \{j \mid k \in \text{scope}(j)\}$$

Instruction k may be executed or not depending on the data tested by the instructions in $\text{dep}(k)$.

3 Standard Bytecode Verification

The bytecode is subject to a static analysis called bytecode verification. A Java bytecode verification algorithm is presented in [11]. Almost all existing bytecode verifiers implement this algorithm. The verification is done on a per method basis and performs a data-flow analysis applied to a type-level abstract interpretation of the virtual machine.

Let us denote by $N_{\text{reg}} \in \mathbb{N}$ the number of local registers required for the method and $N_{\text{stack}} \in \mathbb{N}$ the maximum height that the stack can have during the execution of the method. These two constants can be calculated statically by the compiler.

The types form a domain, where the basic types (e.g. `int`, `address`, $\dots$) are unrelated, $\top$ is the top element. The class types are related as specified by the class hierarchy. In this domain $\top$ represents either the type of an undefined register (not yet assigned) or an incorrect type.

The abstract interpreter executes JVM instructions operating over types instead of values. The verification assigns to each instruction i a mapping M_i from local registers to types and a mapping St_i from the elements in the stack to types. These mappings represent the state $D_i = (M_i, St_i)$ in which the instruction i is performed, thus D_i is the state at the program point i . For each instruction there is a rule that specifies the correct states in which such an instruction can be executed and the state after its execution. For example, an `iload x` instruction requires a non-full stack and the `int` type associated to register x , and its effect is to push `int` onto the stack. Note that a register can have different types at

	pc	current state	D_4	D_8
		M, St	M, St	M, St
0: <code>iconst 0</code>	0	$(1:\top, 2:\top) ()$	$\perp$	$\perp$
1: <code>ifeq 4</code>	1	$(1:\top, 2:\top) (\text{int})$	$\perp$	$\perp$
2: <code>iconst 3</code>	2	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$	$\perp$
3: <code>istore 1</code>	3	$(1:\top, 2:\top) (\text{int})$	$(1:\top, 2:\top) ()$	$\perp$
4: <code>iconst 1</code>	4	$(1:\top = \text{int} \sqcup \top, 2:\top) ()$	$(1:\top, 2:\top) ()$	$\perp$
5: <code>ifeq 8</code>	5	$(1:\top, 2:\top) (\text{int})$	$(1:\top, 2:\top) ()$	$\perp$
6: <code>aconst null</code>	6	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$
7: <code>astore 2</code>	7	$(1:\top, 2:\top) (\text{addr})$	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$
8: <code>iconst 1</code>	8	$(1:\top, 2:\top = \text{addr} \sqcup \top) ()$	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$
9: <code>ireturn</code>	9	$(1:\top, 2:\top) (\text{int})$	$(1:\top, 2:\top) ()$	$(1:\top, 2:\top) ()$

(a)

(b)

Fig. 2. (a) A bytecode (b) An example of verification

different program points, but it must be assured that the state after an instruction must be compatible with the state required by the successor instruction(s): for example, the state after the execution of an `ifeq  $L$` instruction at address i must be compatible with D_{i+1} and D_L .

The rules are used in a standard fixpoint iteration using a worklist algorithm[5]: an instruction i is taken from the worklist and the states at the successor program points are computed. The computed state for a successor program point j is merged with the previous state corresponding to the point j . If the state at j changes, (either the state at j was not yet computed or the already computed state differs), j is added to the worklist. The fixpoint is reached when the worklist becomes empty. Initially, the worklist contains only the first instruction of the bytecode. The initial stack and register types represent the state on method entrance: the stack is empty and the type of the registers corresponding to the parameters are set as specified by the signature of the method. The other registers hold the undefined type $\top$.

As a consequence of the algorithm, the state at a program point of the instructions representing a merge point between control paths, i.e. having more than one predecessor in the control flow graph, is the *least upper bound* of the states after all predecessor instructions. If, for example, register x has type `int` on a path and type $\top$ on another path, the type of x at the merge point is $\top$. The least upper bound of stacks and memories is done pointwise. The pointwise least upper bound between stacks requires that the stacks have the same height. Otherwise there is a type error.

Figure 2(a) shows a simple bytecode method, with no parameters and that returns an `int`. It is accepted by the verifier. Figure 2(b) shows the typing assignment to the instructions produced by the Sun's verifier. In this case, the current state and two dictionary entries are used.

4 The Proposed Verification Algorithm

In this section we propose an approach that exploits immediate postdominators in control flow graphs as a means of reducing the memory required by the verification process.

Instructions are abstractly executed under a context c . This context represents the conditional branches in the control flow graph that affect the instruction we are verifying. In particular, c is represented as a stack that contains an entry for each of the **if** encountered in a path of the graph from the initial node to the instruction and such that its *ipd* is not yet been reached. The main point of the algorithm is that both the alternatives of the open **if** are verified before executing the immediate-postdominator of such branching instruction. During the abstract interpretation, whenever we encounter an **if** instruction at address i , we push information on the given **if** onto c . We push off from this stack whenever we are going to execute an instruction that is the *ipd* of the top of c and all branches have been verified for such **if** instruction. The abstract interpreter maintains a dictionary with an entry for each program point that is either the target or the *ipd* of an open **if**. The dimension of the dictionary changes during the verification process. This allows, in an implementation, to allocate/deallocate dynamically the entries. The insertion of a new dictionary entry D_i in D results into a special state $\perp$ such that, for every $S = (M, St)$ it is: $\perp \sqcup (M, St) = (M, St)$.

In the following, we assume A^* denotes the set of finite sequences of elements of A ; λ indicates the empty sequence and $\cdot$ denotes both the concatenation of a value to a sequence and the concatenation between sequences.

A context is a stack of enriched addresses. An enriched address is a pair (i, n) , where $i \in \mathcal{L}$, $\beta[i] = \mathbf{if}$ and $n \in \mathbb{N}$. Given (i, n) , n represents the number of alternatives of the **if** instruction at address i already verified. We assume $\mathcal{A}$, ranged over by $a, a', \dots$, is the domain of enriched addresses, and $\mathcal{C} = \mathcal{A}^*$ is the domain of contexts, ranged over by $c, c', c_1, \dots$.

The rules of abstract interpreter are reported in Fig. 3. A state Q of the verification has the structure $\langle i, c, S, D \rangle$, where i represents the contents of the program counter, c is the context, $S = (M, St)$ is the current state of registers and stack, and D is the dictionary. The initial state is $\langle 0, \lambda, (M_0, \lambda), \emptyset \rangle$, where M_0 assigns a type to the register according to the method parameters. Note that the verification starts from instruction 0 with an empty context and an empty dictionary D . The rules define a relation $\Rightarrow \subseteq \mathcal{Q} \times \mathcal{Q}$, where $\mathcal{Q}$ is the domain of the verification states.

In the rules, we use the following notation and functions. We assume $ipd((i, n))$ returns the *ipd* of the address field of the pair, i.e. $ipd((i, n)) = ipd(i)$. Similarly, given a context $c \in \mathcal{C}$ and an address i , we assume $i \in c$ denotes the function that returns **true** if a pair (i, n) belongs to c , **false** otherwise.

$$i \in c = \begin{cases} \mathbf{true} & \text{if } c \neq \lambda \wedge \exists j \mid c[j] = (i, n) \\ \mathbf{false} & \text{otherwise} \end{cases}$$

Given a conditional branch instruction i , $target(i)$ returns the target of the branch, i.e. $target(\mathbf{if} L) = L$. Given a context c and an address i , the function

$$\begin{array}{l}
\mathbf{ipd}_0 \quad \frac{c = (j, 0) \cdot c' \quad i = \mathbf{ipd}(j)}{\langle i, c, S, D \rangle \Rightarrow \langle \mathbf{target}(j), (j, 1) \cdot c', D'_{\mathbf{target}(j)}, D' = D[D_i := D_i \sqcup S] \rangle} \\
\mathbf{ipd}_1 \quad \frac{c = (j, 1) \cdot c' \quad i = \mathbf{ipd}(j)}{\langle i, c, S, D \rangle \Rightarrow \langle i, c', D_i \sqcup S, D \downarrow_{c'} \rangle} \\
\mathbf{if}_{\neq} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \mathbf{ifeq} \mathbf{L} \quad i \notin c \quad S = (M, \mathbf{int} \cdot St)}{\langle i, c, S, D \rangle \Rightarrow \langle i + 1, (i, 0) \cdot c, (M, St), (D \uparrow_i [D_L := D_L \sqcup (M, St)]) \rangle} \\
\mathbf{if}_{=} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \mathbf{ifeq} \mathbf{L} \quad i \in c \quad S = (M, \mathbf{int} \cdot St) \quad (M, St) = D_L}{\langle i, c, S, D \rangle \Rightarrow \langle \mathbf{ipd}(\mathbf{top}(c)), c, D_{\mathbf{ipd}(\mathbf{top}(c))}, D \rangle} \\
\mathbf{if}_{\neq} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \mathbf{ifeq} \mathbf{L} \quad i \in c \quad S = (M, \mathbf{int} \cdot St) \quad ((M, St) \sqcup D_L) \neq D_L}{\langle i, c, S, D \rangle \Rightarrow \langle i, c' = \mathbf{extract}(c, i), S, D \downarrow_{c'} \rangle} \\
\mathbf{end} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad i = \mathbf{END} \quad c = \lambda}{\langle i, c, S, D \rangle \Rightarrow \mathbf{success}} \\
\mathbf{load} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \alpha \mathbf{load} \ x \quad S = (M, St) \quad \#St < N_{\mathbf{stack}} \quad M(x) = \alpha}{\langle i, c, S, D \rangle \Rightarrow \langle i + 1, c, (M, \alpha \cdot St), D \rangle} \\
\mathbf{const} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \alpha \mathbf{const} \ x \quad S = (M, St) \quad \#St < N_{\mathbf{stack}}}{\langle i, c, S, D \rangle \Rightarrow \langle i + 1, c, (M, \alpha \cdot St), D \rangle} \\
\mathbf{store} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \alpha \mathbf{store} \ x \quad S = (M, \alpha \cdot St')}{\langle i, c, S, D \rangle \Rightarrow \langle i + 1, c, (M[x := \alpha], St'), D \rangle} \\
\mathbf{return} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \alpha \mathbf{return} \quad S = (M, \alpha \cdot St')}{\langle i, c, S, D \rangle \Rightarrow \langle \mathbf{END}, c, (M, St'), D \rangle} \\
\mathbf{goto} \quad \frac{i \neq \mathbf{ipd}(\mathbf{top}(c)) \quad \beta[i] = \mathbf{goto} \ \mathbf{L}}{\langle i, c, S, D \rangle \Rightarrow \langle \mathbf{L}, c, S, D \rangle}
\end{array}$$

Fig. 3. Rules of the abstract interpreter

$\mathbf{extract}(c, i)$ pops from c the topmost elements until (i, n) included. Let $c = c' \cdot (i, n) \cdot c''$, $\mathbf{extract}(c, i)$ returns c'' .

Given a dictionary D and a context c , we use $D \downarrow_c$ to denote the restriction of D to the before state of instruction that are either target or $\mathbf{ipd}$ of an **if** that is on the context c . Therefore, $D \downarrow_c$ removes from D all the dictionary entries that are no longer needed when the context is c .

That is $D \downarrow_c = D - \{D_j \mid \exists i \in c, j = \mathbf{ipd}(i) \vee j = \mathbf{target}(i)\}$.

Moreover, let a branching instruction i , we use $D \uparrow_i$ to denote a new dictionary D' obtained from D by adding the entries $\mathbf{target}(i)$ and $\mathbf{ipd}(i)$ if not present in D . Finally $D[D_i := S]$ denotes the dictionary D' which agrees with D on all entries except D_i for which it is $D'_i = S$. The same notation is used for memory M .

The rules for **ipd** are applied before of any other rule (see the premise of the rules). When the verification reaches the instruction $i = \mathbf{ipd}(j)$ of the j : **if** topmost element of c , we distinguish two cases depending on whether both alternatives of j have already been verified or not. Rule **ipd**₁ or Rule **ipd**₀ are applied, respectively.

Rule **ipd**₁. The **if** on top of c has been verified. The topmost element of c is popped and execution continues starting from j . The current state is merged

with the state saved in D for i ; because the context is changed, D is updated removing all its unnecessary entries.

Rule **ipd₀**. The number of verified alternatives for the **if** is incremented and the dictionary entry D_i is updated to the least upper bound between D_i and the inferred type for the instruction. Then the other alternative of j is executed.

There are different rules for **if**, applied depending on the context c . In presence of cycles, an **if** instruction can be reached many times. An **i: ifeq L** instruction is inserted onto c only if i is not an *open if*.

Rule **if_≠**. Any time an **i: ifeq L** instruction is reached and the given **if** is not open, the information on the **if** are pushed onto the stack c : i is the address of the **if**; the number of alternatives of **if** already verified is set to 0. If necessary, the entry for $target(i)$ and $ipd(i)$ are added to the dictionary. Moreover, the entry for the target is merged with the after state of i .

Any time the **if** reached is open, two different rules can be applied.

Rule **if₌**. The inferred type for i is a subtype of the one previously saved in D_i . In this case, we must not re-verify the program from this point. Verification continues from the ipd of the top of c . The rule for **ipd** increments the number of alternatives already verified of the topmost element of c .

Rule **if_≠**. The inferred state for i is not a subtype of the one previously saved in D_i . In this case, as the state of the **if** has changed, we have to restart the verification from there. The instruction at address i must be re-verified starting from a different register and stack type. Assume j is the position in c of the pair (i, n) , all the elements on c from j (included) to the top are popped off the stack. By construction, $\forall i$ there is at most one pair with address equal to i in c .

Because the program counter is not changed, the next rule to apply will be Rule **if_≠**.

The other rules are applied for instructions i different from **if** and that are not ipd of an **if** in c . They simply update the current state with the after state of i (as in the standard bytecode verification), and set the program counter to the unique successor of i .

When a **return** instruction is reached, execution continues with the program counter equal to **END**. The rule for **end** is applied when $i = \text{END}$ and the context is empty. This represents the success of the verification. In the case in which $i = \text{END}$ and the context is not empty, **END** is the ipd of the **if** on top of c . By definition of ipd , the node **END** is the ipd of every other **if** onto the context. Verification continues by applying the **ipd** and the **end** rules.

Note that the context is finite, since the Rules for **if** assure that $\forall i$ there is at most one pair with address equal to i in the context.

Consider a control flow graph such that the final node is reachable from any other node. The following lemma assures that ipd are reached along every execution path in the reverse order with respect to the corresponding conditional instructions.

Rule to apply	pc	context c	current state $M \ St$	Dictionary D
const	0	λ	$(1:\top, 2:\top) \ ()$	$\emptyset$
if_g	1	λ	$(1:\top, 2:\top) \ (\text{int})$	$\emptyset$
const	2	$(1, 0)$	$(1:\top, 2:\top) \ ()$	$4:(1:\top, 2:\top) \ ()$
store	3	$(1, 0)$	$(1:\top, 2:\top) \ (\text{int})$	$4:(1:\top, 2:\top) \ ()$
ipd₀	4	$(1, 0)$	$(1:\text{int}, 2:\top) \ ()$	$4:(1:\top, 2:\top) \ ()$
ipd₁	4	$(1, 1)$	$(1:\top, 2:\top) \ ()$	$4:(1:\top = \top \sqcup \text{int}, 2:\top) \ ()$
const	4	λ	$(1:\top, 2:\top) \ ()$	$\emptyset$
if_g	5	λ	$(1:\top, 2:\top) \ (\text{int})$	$\emptyset$
const	6	$(5, 0)$	$(1:\top, 2:\top) \ ()$	$8:(1:\top, 2:\top) \ ()$
store	7	$(5, 0)$	$(1:\top, 2:\top) \ (\text{addr})$	$4:(1:\top, 2:\top) \ ()$
ipd₀	8	$(5, 0)$	$(1:\top, 2:\text{addr}) \ ()$	$4:(1:\top, 2:\top) \ ()$
ipd₁	8	$(5, 1)$	$(1:\top, 2:\top) \ ()$	$4:(1:\top, 2:\top = \top \sqcup \text{addr}) \ ()$
const	8	λ	$(1:\top, 2:\top) \ ()$	$\emptyset$
return	9	λ	$(1:\top, 2:\top) \ (\text{int})$	$\emptyset$
end	END	λ	$(1:\top, 2:\top) \ ()$	$\emptyset$
success				

Fig. 4. An example of verification using our approach

Lemma 1. *Let i, j two nodes of the control flow graph such that $\beta[i] = \text{if}$ and $\beta[j] = \text{if}$. Suppose that there is a path from node i to node j not containing $\text{ipd}(i)$. Assume $\text{ipd}(i) \neq \text{ipd}(j)$. It holds that $\text{ipd}(i) \text{ pd } \text{ipd}(j)$.* $\square$

Lemma 1 justifies the stack data structure of the context. Moreover, in the rules, the *ipd* to be considered is the *ipd* of the open **if** instruction on top of the context.

Figure 4 shows the verification algorithm applied to the example in Figure 2(a). In this case the dictionary contains at most one entry. In the considered example, for each **if**, the *ipd* coincides with the branch target. Applying the algorithm, and starting from instruction 0, these relevant actions are executed:

- A dictionary entry is inserted in D for the instruction 4 (since 4 is both the branch target and *ipd* of the **ifeq** at label 1)
- The two branches are executed abstractly, and then the resulting states are merged when the *ipd* is reached.
- Since all alternatives have been verified, before allocating a new entry in the dictionary for the **ifeq** at label 5, we can successfully remove the one for the **ifeq** at label 1.

Theorem 1. *The verification algorithm applied to a bytecode β terminates with success if and only if β is accepted by the standard Verifier.* $\square$

4.1 Memory Requirements

The following theorem gives the upper bound to the dimension of the dictionary during the verification process.

Theorem 2. *The maximum number of elements in D during the verification of bytecode β is:*

$$\max \left\{ \# \bigcup_{i \in \text{dep}(k)} \{ \text{target}(i) \cup \text{ipd}(i) \}, \forall k \in \mathcal{L}(\beta) \right\}. \quad \square$$

Let us denote by m the maximum nesting level of the conditional instruction in the bytecode: $m = \max \{ \# \text{dep}(k) \mid k \in \mathcal{L}(\beta) \}$. The maximum length of the context is bounded by m . Because for each context item at most two elements are added to D , then $\#D \leq 2m$. Since often targets and *ipds* coincide, then generally $n < 2m$.

The complexity of our algorithm in terms of space is $O(n * (N_{\text{stack}} + N_{\text{reg}}))$, since the space requested by the other data structures is negligible compared to that of the dictionary. The Sun's Verifier instead, has a complexity in space equal to: $O(N_{\text{branch}} * (N_{\text{stack}} + N_{\text{reg}}))$, where N_{branch} is the number of distinct branch targets in the method.

The reduction of memory is significant when the nesting level m of the conditional instructions in the code is less than the total number of branch targets. As an example, considering the **Purse** and **Loyalty** sample applets included in the Java Card Kit [8], m is, on average, an half the number of branching instructions in the code. However, the worst case must be taken into account and the maximum memory space is required for long methods. Empirically, longer is the method body, less is the ratio between m and the total number of branches, thus providing a reasonable reduction of the memory used. The disadvantage of our approach is the execution time. Execution time can be worst with respect to the time taken by the standard verification process, since some part of the bytecode can be verified more than once also for the same type inferred for an instruction (but with different contexts). The cost in time of the proposed algorithm is under study, since a prototype tool is under development.

5 Related Work and Conclusions

The goal to find a practical solution for implementing a bytecode verification under the severe constraint imposed by small computers (as smart card) associates many research works.

A cryptography-based approach relies on the fact that applet issuer sends the **capfile** to a trusted third party (TTP) that digitally signs it with his private key. Who wants to install the applet has to check, using TTP public key, that downloaded code is identical to the one code producer sent to the TTP. This mechanism assumes trust relations between code producer and code consumer and between code consumer and TTP exist.

To overcome the limitations of the cryptographic solution, Rose and Rose [10] proposed to split the verification in two phases: lightweight bytecode certification (LBC) and lightweight bytecode verification (LBV). LBC, to be performed off card, produces a certificate (that must be distributed with the bytecode). LBV (on card) is a verification that uses both bytecode and certificate to assure that bytecode is safe. Rose and Rose, that were inspired by the PCC, "proof carrying

code” work by Necula [9], show that LBV is completely equivalent to standard bytecode verification, proving that LB can only succeed if and only if bytecode verification succeed, and that is not possible to build a certificate that can bring the LBC to accept non-valid bytecode. Lightweight bytecode verification is currently used in the KVM of Sun’s Java 2 Micro Edition.

Leroy, in [6] shows that, imposing that operand stack must be empty at branch points (*stack normalization*) and that registers have only one type throughout the method, it is possible to optimize the verification in terms of memory requirements. However, this approach leads to a growth in the number of registers used, and because bytecode could not satisfy the requirements above, it needs a pre-transformation. In this way the verification is done on card, but it could refuse safe code that has not be pre-processed.

Deville and Grimaud [2] states that, storing the dictionary not in RAM but in persistent memory (i.e. EEPROM) and a particular non-stressing type encoding to speed up writes, a stand-alone verifier can be implemented integrated on a smart card.

In this paper the memory problem is attacked by using a dictionary whose number of elements changes during the verification. The live range of an entry is calculated by the control dependencies of the instructions and uses the notion of immediate postdominator in control flow graphs. A prototype of a Verifier based on the proposed approach is under development. The tool reuses the library BCEL of the Jakarta Project [3] that contains a Java source code that implements the standard bytecode Verifier. Further work is to extend our approach to include subroutines. We use the polyvariant data flow analyses of subroutines, as implemented in Java-card off card verifiers [7]. A stack, called *contour*, that maintains the return address of the nested `jsr` instruction encountered along the execution path is used and the subroutine body is analyzed several times. Our bytecode verification algorithm can be modified by treating a `i: jsr L` instruction as an unconditional branch to `L` and a `i: ret x` instruction as an unconditional branch to the topmost element onto the contour. The control flow graph can be extended to include subroutines body as follows: an edge from node `i : jsr L` to node `i+1` is added ($\forall i : \text{jsr } L, (i, i + 1) \in V$); moreover, no edges are inserted starting from node `ret` to any other node ($\forall i : \text{ret}, \nexists j \mid (i, j) \in V$). The subroutine body is then verified similarly to the bytecode of the method body.

References

1. T. Ball. What’s in a region? or computing control dependence regions in near-linear time for reducible control flow. *ACM Letters on Program. Lang. Syst.*, 2(1–4):1–16, 1993.
2. Damien Deville and Gilles Grimaud. Building an “impossible” verifier on a Java Card, 2002.
3. Apache Foundation. Jakarta project, <http://jakarta.apache.org/index.html>.

4. J. A. Gosling. Java intermediate bytecodes. In *ACM SIGPLAN Workshop on Intermediate Representations Proceedings*, pages 111–118. ACM, 1995.
5. G. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st Annual ACM Symposium on Principles of Programming Languages*, pages 194–206, 1973.
6. X. Leroy. Bytecode verification for java smart card. *Software Practice & Experience*, 32:319–340, 2002.
7. X. Leroy. Java bytecode verification: algorithms and formalizations. *Journal of Automated Reasoning*, 2003. To appear in the special issue on Java bytecode verification.
8. Sun Microsystem. Java Card 2.2 Development Kit.
9. G. C. Necula. Proof-carrying code. In *24th Annual Symposium on Principles of Programming Languages Proceedings*, pages 106–119, January 1997.
10. E. Rose and K. Rose. Lightweight bytecode verification. In *WFUJ 98 Proceedings*, 1998.
11. Lindholm T. and F. Yellin. *The Java virtual machine specification*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1996.
12. F. Yellin. Low level security in java. In *4th International World Wide Web Conference Proceedings*, pages 369–379. O’Reilly, 1995.

Computational Reflection for Embedded Java Systems

Antonella Di Stefano¹, Marco Fargetta¹, and Emiliano Tramontana²

¹ Dipartimento di Ingegneria Informatica e delle Telecomunicazioni,
Catania University, Viale A. Doria, 6 - 95125 Catania, Italy,
`{adistefa,mfargetta}@diit.unict.it`,

² Dipartimento di Matematica e Informatica,
Catania University, Viale A. Doria, 6 - 95125 Catania, Italy,
`tramontana@dmf.unict.it`

Abstract. Although Java reduces the time to market of embedded systems, for some contexts developers are still forced to consider, beside application concerns, checks and handling activities for anomalous conditions that can occur on hardware devices. Typically, applications and handling of anomalous conditions are unrelated, and developers should be provided with means to treat them separately. Reflective systems have been successfully used to separate different concerns. However, special care is required when using them in embedded systems, due to the run time overhead that they can cause. In this paper, we propose what we call selective reflective behaviour, which aims at reducing the run time overhead of reflective systems. An efficient implementation of this behaviour is also proposed, which is suitable even for embedded Java systems. The paper also presents an example of a meta level that handles anomalous conditions for the embedded systems in a production cell.

1 Introduction

In the context of embedded systems, the interest for Java technology continuously increases, and today some embedded systems available include a JVM for running Java applications. This approach allows a fast software engineering process for increasingly complex functionalities that these systems are equipped with.

Many Java features explain its diffusion in embedded systems [6]. The first and the most important of these features is that Java limits the number of errors when developing applications. This goal is achieved since it is a strongly typed object-oriented language and by removing pointers. As a second feature, the code developed for one platform can be easily transferred to others, without any change, thanks to the portability of Java.

Although there are many projects that use Java in embedded systems, inserting a full JVM in a small device where there is not much memory nor high CPU performance is complicated, and different approaches have been followed to cope with these shortcomings. Sun Microsystems experienced that a standard

JVM was inadequate and developed several Java API subsets to be adopted for embedded systems [11]. These subsets are: Personal Java for big embedded systems, J2ME (Java 2 Micro Edition) for embedded systems and KVM (Kilo Virtual Machine) for very small embedded systems. Sun Microsystems also developed a special CPU, named PicoJava [12], which is able to execute only Java bytecode (without interpretation). A very efficient alternative is to compile Java applications into native code with a tool such as gcj [5], allowing them to execute without a JVM¹. However, gcj does not support Java run time class loading.

Many embedded systems work in an environment where the same operations execute continuously, except for a fault or other sporadic operations that alter natural execution. This is frequent in control systems and in other devices used e.g. in a production cell for aided manufacturing where there are only a few operations to do, however there could be many exceptional situations. Since there are many possible faults, the code for handling faults may be bigger than the code for normal operations. This code could even be bigger than the available memory. Moreover, inserting into an application every check for exceptional events makes the application code more intricate and long. We propose a *reflective approach* to reduce the memory used by Java applications for embedded systems and avoid mixing application related and fault handling concerns into a class. Reflection overcomes these problems, since it allows the code handling faults to be separated from application classes, so that each can be loaded selectively, only when needed.

With a reflective approach, developers can structure systems by means of two levels [8,4] (see section 2.1), one for the application and another one for concerns such as faults handling [10], distribution [3], synchronisation [14], etc. In our approach, we separate applications from the code handling faults related with hardware, such as memory overflow, wrong positions of robotic arms, overheated physical devices, etc. This fault handling code can be considered independent of application concerns and can be developed separately. Moreover, reflection allows applications for embedded systems to be customised with new functionalities or with the handling of exceptional conditions (either with an ad-hoc portion of code, developed by the application designer, or with code that normally equips embedded systems). As a further contribution we describe a novel reflective mechanism, called *Selective Reflective Behaviour*, that we have developed to reduce the overhead caused by reflective systems. This is especially useful for embedded systems where CPUs are usually not very powerful, so their time should not be wasted. By selectively trapping control from the application, the overhead of reflective systems is paid only when some fault occurs. As explained in section 5, this overhead amounts to two methods invocations (one is used to determine which class can solve the fault and another to recover the state by performing some operations) and a class loading (when the fault happens for the first time).

¹ This is helpful when real time features need to be inserted into the application since it is only the operating system that handles them. This is a better solution than submitting bytecode to the JVM where an additional scheduler is used.

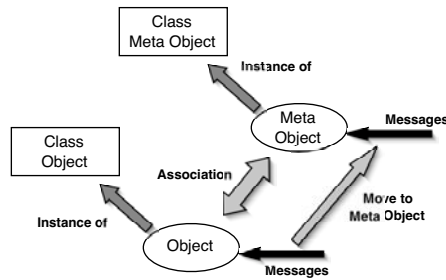


Fig. 1. Meta Object model

This paper is structured as follows. Next section introduces reflection and proposes a new approach for reducing its overhead. Section 3 describes the possible work of a meta level for embedded systems. Section 4 expresses how we have implemented a modified JVM supporting reflection. Section 5 shows a case study that we have developed using our modified JVM. Finally, section 6 draws some conclusions.

2 Supporting Reflection Efficiently

2.1 Reflection

Reflection is a programming paradigm whereby a system contains structures allowing to observe and control itself [8,4]. A reflective system can be represented by two levels: base level and meta level. The base level usually consists of an application, whereas the meta level holds the code that enriches (in some way) the application. The behaviour of object-oriented systems can be observed and controlled by means of *interception* and *inspection* of methods and fields. Interception is the mechanism used by reflective systems to capture control when e.g. methods of an object are invoked or fields are accessed. Inspection is the ability to check the state and structure of objects and classes. The twofold ability of intercepting and inspecting is called *reification* [1].

Reflection can be characterised by several models. In the *Meta Object* model, which is the one we use, application objects are associated with meta level objects, which are object instances of a special class called *Meta Object* or of its subclasses (see Fig. 1) able to intercept messages and so gain control before the associated application objects [8,4].

Some reflective Java extensions (e.g. Javassist [2], Kava [15]) use bytecode manipulation, either off-line or at load-time, to insert some statements into application classes and so allowing control to jump to the meta level when some operations are performed (e.g. a method of a class is invoked, an object is instantiated, etc.). These implementations impose a certain amount of performance degradation due to the handling of the meta level (bytecode manipulation, instantiation of meta objects, jumps to the meta level, etc.). When a meta object

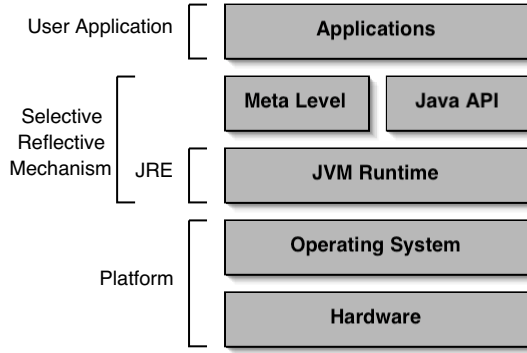


Fig. 2. JVM architecture with the selective reflective behaviour

has been associated with an object, it intercepts the operations performed on the object, causing control to jump to the meta level. As a consequence, interception slows execution down and especially when, due to system conditions, jumping to the meta level is useless this is considered an important drawback. This overhead would be reduced if jumps to the meta level were only performed when the work of the meta level was considered useful.

2.2 The Selective Reflective Behaviour

As said before, in the Meta Object model, messages sent to an application object are intercepted and redirected to the associated meta object. Then messages go from this meta object to an application object (the one invoked originally or another one) or to an appropriate meta level object. Passing control to several objects introduces some overhead.

A solution that we propose to limit this kind of overhead for reflective systems is what we call *Selective Reflective Behaviour*, which allows checking some conditions before trapping control to the meta level. We have achieved this selective reflective behaviour by modifying the version 1.2.2 of the Sun JVM² (see Fig. 2). The JVM is then able to decide when it is the case to enable interception, thus limiting the number of jumps to the meta level and so the overhead of reflective systems. This makes it possible to use reflective systems even in embedded environments, where CPU time cannot be wasted.

This selective reflective behaviour is useful for any reflective system, since it provides designers with means to set the degree by which interceptions have to be performed. As explained in the following section, designers communicate to the JVM when to intercept events by means of a file or a class annotation.

For embedded systems, we propose to use the meta level to handle hardware faults, thus, in normal conditions, the jump to the meta level is unnecessary. In

² An alternative implementation that we have produced to achieve this selective behaviour only modifies a JIT and is based on OpenJIT [9].

this case, having a selective reflective behaviour is very useful, since in normal conditions the JVM only invokes base level methods, avoiding the jump to the meta level. This reduces the overhead by one method call and the reflective system can be considered efficient for embedded systems. On the other hand, when some events move the system from its normal conditions, jumps to the meta level are enabled by the JVM. As far as memory use is concerned, in our approach application classes are smaller, since they do not implement any fault handling, which is instead implemented as meta level classes. These latter classes are only loaded into main memory when some fault has occurred and therefore in normal conditions the amount of memory used diminishes.

When developing software for embedded systems using the proposed approach, we can distinguish an application and a system designer. The application designer provides application classes implementing only those operations that have to be performed at run time, assuming that the execution does not meet anomalous conditions. The system designer provides the meta level classes implementing activities, unrelated to one particular application, that are necessary to detect anomalous conditions and to recover the system from such conditions. The system designer therefore specifies the conditions (for system variables) for which jumping to the meta level is necessary. The application designer can provide additional conditions (typically depending on application variables) that when satisfied determine a jump to the meta level. Moreover, the application designer can implement new meta level classes handling faults that were not considered by the system designer.³

Our use of the reflective mechanism could appear similar to the exception handling mechanism, since when a fault or other exceptional events occur the JVM executes a code handling the exception, however there are at least two main differences. The exception handling mechanism is used inside the application code by means of a **try-catch** block, instead when using the reflection mechanism the application code is not modified, and all the operations handling exceptional events are inserted into the meta level, which is detached from the application. Therefore, the reflective mechanism allows applications to be customised without being intrusive. Moreover, the reflection mechanism allows designing smaller classes, since the code handling exceptions is outside them and thus application classes for normal conditions take less time to be loaded and a smaller amount of memory once loaded. This is important for embedded systems where performance and memory are limited.

3 The Reflective Behaviour for Embedded Systems

3.1 Work of the Meta Level

Within the proposed meta level, the first operation would be to understand the conditions that caused interception. As Fig. 3 shows, this is achieved by

³ Details on the meta level classes and their connection with application classes can be found in section 5.

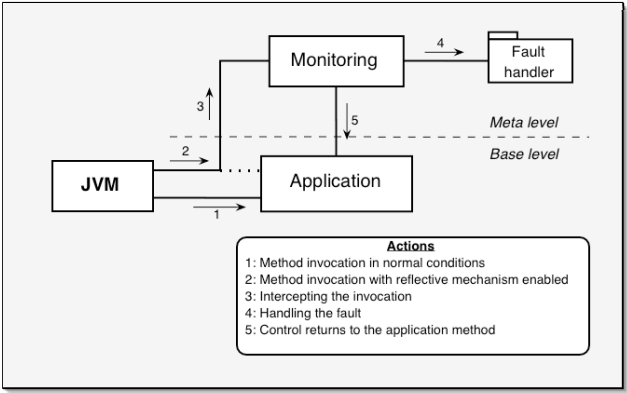


Fig. 3. Selective reflective behaviour

a **Monitoring** meta object that checks the trapped object operation, the state of the application and hardware. The check aims at determining how the fault has to be handled and so the object that should receive control to recover the system. We expect the related class to be loaded by the JVM from the local file system, however it can also be downloaded from the network. Once this class is determined, an instance is created and control is passed to it. The mapping between anomalous condition and handling class is controlled by meta object **Monitoring** and a class library, on the basis of checks given by the meta level designer and of additional conditions that the application designer provides.

As an example, let us suppose that in a production cell a robotic arm has reached a bad position, then as soon as the JVM receives control, it detects such a condition and sends control to the meta level. The recovering code is implemented as meta level class **RecoveryArm** that meta object **Monitoring** knows and to which control is passed.

3.2 Enabling Interception

The selective reflective mechanism is activated whenever the JVM detects some anomalous condition. The description of the check that reveals anomalous conditions is inserted inside the JVM, however the application programmer can provide an additional check list, by means of a XML file. This file, located in the directory where application classes are found, describes: the conditions for which the meta level is allowed to trap control, the meta objects that are associated with application classes, and the operations that meta objects should trap.

In the XML file, we use the tag **<MetaClass>** to specify the name of the meta object class. The inner tag **<BaseClass>** allows the meta level class to be associated with an application class. Analogously, the tag **<Interception>** is used to specify which fields and methods have to be intercepted; and the tag

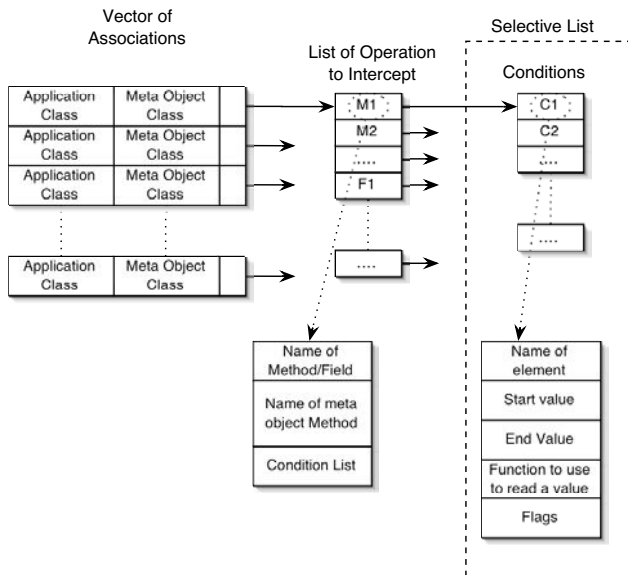


Fig. 4. Mapping data structure

<Condition> allows expressing the conditions that enable the meta level to trap control from the application (section 5 shows an example of such a file).

Being the XML specification separated from application and meta level classes, it can be easily modified to allow new connections between base level and meta level classes.

4 Modifying the JVM

The JVM needs extra information to support the selective reflective behaviour, in order to choose the appropriate object to give control to at run time. This extra information is organised as a list of system variables or application fields, together with the values to be checked when a switch to the meta level has to be enabled. Such a list, called **selective list**, includes system variables known by the modified JVM and information taken from a file.

As Fig. 4 shows, the **selective list** is part of a mapping data structure that allows the JVM to know for each class the corresponding meta object and the operations to be trapped. Such a structure consists of: an **association vector** that contains the association between application classes and their meta object classes; a **list of operations** that have to be intercepted; the **selective list** containing those conditions that have to be true to allow interception.

Because of its internal functioning, the JVM performs the check that may allow jumping to the meta level quickly. When an application carries out a field access or a method invocation (see lines from 4 to 17 in listing 4), the JVM

searches for the location of the object in heap memory. At this point verifying the values of the fields enabling interception is very simple and fast. The list of conditions is then retrieved and if one condition is verified the method invocation is substituted (see lines from 25 to 33 in listing 4). This substitution is easily performed since the JVM holds the references to application objects and meta level objects. It is only necessary to prepare the parameters of the method that is actually called⁴.

Moreover, values inside the hardware system (temperature sensors, contact sensor, etc.), or originated from the processing hardware (e.g. memory busy), are easily accessed by modifying some parts of the JVM. Generally, JVMs are implemented in C or C++ and by exploiting the features of these low level languages, they can be modified to achieve fast access to the state of some hardware devices. On the other hand, as Fig. 2 shows, Java applications cannot see the operating system nor the hardware directly and so access to the underlying hardware would be slow and difficult (except when an application uses JNI [13]).

4.1 Benefits When Modifying the JVM

The implementation of the Selective Reflective Behaviour can be efficiently achieved by either modifying the JVM or the JIT. We have produced a modified JVM, since the small amount of memory and the low CPU performance available in embedded systems makes it inadequate to use a JIT.

Some authors have proposed to use a JIT for embedded systems [7]. With such an approach instead of modifying the whole JVM, it would be possible to have a JIT that provides an application with the selective reflective behaviour. This allows the JIT to be completely removed when unnecessary, just by setting an environment variable, thus avoiding to perform any additional check for those applications that need so. We have implemented a modified version of OpenJIT that provides applications with selective reflective behaviour. However, we think that it is better to integrate this into JVMs for PCs.

By modifying a JVM to include reflective abilities, some significant benefits are achieved with respect to other approaches. Firstly, for the set up, the JVM needs only to instantiate some structures handling references to meta level classes and objects. The timeframe for such a set up is much smaller than the time needed to modify a class bytecode, which has to be performed each time a class is loaded (this approach is adopted by Kava and Javassist). Secondly, at runtime, switching to the metalevel is very fast, since control goes to the JVM whenever an application object invokes methods or access fields. Inside the JVM, executing a check to pass control to the associated meta object is faster than having two invocations (one for the application object and one for the meta object) as it happens with Kava and Javassist.

Finally, an ad-hoc version of the JVM could be installed into embedded systems to enrich them with a better support. Having a non-standard JVM

⁴ The meta level method intercepting the call has a different list of parameters than the trapped method.

Listings 1.1. Modifications to `ExecuteJava`

```

1  bool_t ExecuteJava(unsigned char *initial_pc , ExecEnv *ee) {
2      // .....
3      switch (opcode) {
4      case opc_invokevirtual:
5          // Select the object whose method has to be called
6          DO_INVOKER(FALSE);
7          if (opcode == opc_invokevirtual) {
8              // Select the method block and set program counter, etc.
9              if (mb->fb.u.offset != 0)
10                 mb = mt_slot(obj_array_methodtable(o), mb->fb.u.offset);
11                 frame->returnpc = 0;
12                 optop -= args_size;
13                 // Jump to the code where the invocation is performed
14                 goto callmethod;
15             } else {
16                 // Restore the stack
17                 SIZE_AND_STACK(0, 0);
18             }
19         // .....
20         callmethod:
21         sysAssert(o != 0);
22         DECACHE_OPTOP();
23         {
24             // Retrieve the conditions whereby methods are called
25             lst_condition = retrieveCondition(mb);
26             // Check the list of the conditions
27             while (lst_condition != NULL)
28                 if (check(lst, o, mb)) {
29                     // If this check is true the method invoked is
30                     // the one at the meta level
31                     mb = changeMethod(lst, mb, o);
32                     o = changeObject(lst, mb, o);
33                     break;
34                 }
35             // Method invocation
36             bool_t result = mb->invoker(o, mb, args_size, ee);
37             // .....
38         }

```

should not be considered as a problem for these systems, since it is not expected to have a previous JVM installed with configured libraries, etc. nor to have applications that could be harmed by the inserted features (since applications for these embedded systems are known, compatibility tests can be run a priori). Moreover, applications are developed without using any feature of the modified JVM, in fact they are unaware of modifications, thus they would properly work on any JVM, given that faults are handled in another way.

5 Case Study

After that we have described how the selective reflective behaviour works, we use it for a case study in order to show how it can help the development of applications for embedded systems.

In modern factories there are assembly lines that are fully managed by robots. Each robot could be considered as an embedded system with its operating system and the application that controls its movements. The work of the robot is repetitive but many faults can occur and a software portion should handle them. The selective reflective behaviour is useful to simplify the development and maintenance of this software portion, thanks to the separation it enables between application and faults handling concerns.

Let us consider an application consisting of some classes that control the robot actions. The method classes are developed without needing to check the conditions of the environment or the application itself, instead this work is delegated to the meta level. In normal condition, i.e. when no problems occur to position arms, etc., the meta level is disabled and the application executes as expected. When these conditions change it is necessary to execute some code that handles the abnormal state and avoids malfunctioning.

In our example application, class **Actions** is dedicated to position the robot and includes methods: **moveArm()**, to control the arm movements; and **changePosition()**, to move the robot inside the production cell. These methods use two fields, **armPos** and **robotPos**, that hold the coordinates of arm and robot, respectively.

To handle the problems arising in exceptional conditions, a meta level is connected to the application. This meta level holds a class **Monitoring** that obtains control from the JVM when an exceptional condition is recognised. The **Monitoring** class checks the state conditions and forwards a request to a repository server specifying an identifier for the fault. The repository server returns a class able to handle the fault (see Fig. 3).

Listing 5 shows class **Monitoring**. This class implements the **MetaObject** interface that consists of two methods, **trapField()** and **trapMethod()**, which respectively intercept field access and method invocation. Two parameters are passed to **trapField()**, which identify the name of the field and the reference to the object from which control has been trapped. Method **trapMethod()** has two parameters, which determine the name of the trapped method and an array of objects, whose first element is the trapped object. The other elements of this array are the parameters passed to the original method.

To activate the selective reflective mechanism a file with the conditions to be checked is written. As shown in listing 5, in this file there could be different types of conditions enabling interception. Some conditions are used to check the arm position (expressed by means of tag **<range>**), which is given by a value inside the application class (specified by means of tag **<nameF>**). Other conditions refer to the memory and the CPU workload (expressed by means of tags **<memoryFree>** and **<workload>**, respectively). The first and second condition enable a field access to be intercepted, since they are specified within tag **<field>**, whereas

Listings 1.2. Meta level class `Monitoring`

```

1  class Monitoring implements MetaObject {
2      Class cls;
3      FaultIndex fidx;
4      Object state [];
5      public Monitoring(Object o[]) {
6          cls = o[0].getClass();
7          // Retrieve the list of fault identifiers
8          fidx = Repository.getFaultIndex(cls);
9          // Create a checkpoint for the object
10         storeState(o);
11     }
12     public void trapMethod(String name, Object o[]) {
13         Handle hand;
14         FaultGen fg;
15         // Detect the conditions that have generated the fault
16         fg = checkFault(name, o);
17         for(int i=0; i<fidx.length; i++)
18             // Verify if this fault identifier is appropriate
19             if (fg.compare(fidx[i]) {
20                 // Obtain handler from Repository and start execution
21                 hand = Repository.getHandler(fidx[i]);
22                 hand.startHandler(name, o);
23             }
24         return;
25     }
26     public void trapField(String name, Object o) {
27         //....
28     }
29 }

```

the third condition enables intercepting a method call, since it is specified within tag `<method>`.

When the application executes, if one condition is true, e.g. the CPU workload for the embedded system is over 30 processes in the run queue, the JVM on invocation of method `moveArm()` enables the reflective behaviour and provides control to `trapMethod()` of the `Monitoring` meta object. This meta object analyses the system conditions and invokes the `Repository` to obtain a class to handle the fault. Then control returns to the application and the meta level is disabled until a condition is true again.

6 Conclusions

The approach that allows reflection for embedded systems that we have proposed here provides at least three benefits. Firstly, application classes are not forced to include code to handle exceptional events, thus their development is easier

Listings 1.3. XML configuration file

```

1 <?xml version="1.0"?>
2 <MetaLevel>
3   <MetaClass>
4     <nameMO>Monitoring</nameMO>
5     <BaseClass>
6       <nameBO>Actions</nameBO>
7       <Interception>
8         <field>
9           <nameF>arm</nameF>
10          <nameFInterceptor>trapField</nameFInterceptor>
11          <Condition>
12            <range>
13              <startValue>arm.x=30</startValue>
14              <endValue>arm.x=90</endValue>
15            </range>
16            <!-- Enable interception for some arm values -->
17            <memoryFree>1M</memoryFree>
18            <!-- Enable interception if free memory is
19                  less than 1MB-->
20          </Condition>
21        </field>
22        <method>
23          <nameM>moveArm</nameM>
24          <nameMInterceptor>trapMethod</nameMInterceptor>
25          <Condition>
26            <workload>30</workload>
27            <!-- Enable interception if workload is
28                  over 30 processes -->
29          <Sensor>
30            <nameS>TemperatureSignal</nameS>
31            <range>
32              <startValue>50</startValue>
33            </range>
34            <!-- Enable interception if temperature is
35                  over 50 C-->
36          </Sensor>
37        </Condition>
38      </method>
39    </Interception>
40  </BaseClass>
41 </MetaClass>
42 </MetaLevel>

```

than that in traditional approaches. Secondly, when the application is moved into a new environment, it does not need to be modified to insert the code handling possible faults. This allows moving an application in a new system in a shorter time, whereas with the traditional approach, applications have to be

reengineered to insert checks. Thirdly, when a class handling a fault condition has to be changed, only the repository needs to be updated and the embedded system could execute without stopping. This simplifies updating an application and minimises downtime of the system.

The engineering approach that we have proposed reduces the effort when developing applications. We have achieved this by separating applications and fault handling by means of a novel reflective model. This separation can be considered similar to the one found in Aspect Oriented Programming, where components (which in our case are application classes) and aspects (code handling faults) are developed separately and then connected by an ad-hoc compiler called weaver. However, for the requirements of embedded systems, we needed to selectively enable at run time the code handling faults. This is not possible to achieve by simply using aspects that are always active. Moreover, we have lowered the memory used by the application by extracting faults handling from classes. The reflective mechanism ensures that this holds at run time, whereas what the run time condition of the memory is when using aspects depend on the implementation choices of the weaver used.

References

1. W. Cazzola. Evaluation of Object-Oriented Reflective Models. In *Proceedings of the ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*, Brussels, Belgium, 1998.
2. S. Chiba. Load-time Structural Reflection in Java. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'00)*, volume 1850 of *Lecture Notes in Computer Science*, 2000.
3. A. Di Stefano, G. Pappalardo, and E. Tramontana. Introducing Distribution into Applications: a Reflective Approach for Transparency and Dynamic Fine-Grained Object Allocation. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC'02)*, Taormina, Italy, 2002.
4. J. Ferber. Computational Reflection in Class Based Object Oriented Languages. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89)*, volume 24 of *Sigplan Notices*, pages 317–326, New York, NY, 1989.
5. Free Software Foundation, <http://gcc.gnu.org/onlinedocs/gcj.ps.gz>. *Guide to GNU gcj*.
6. V. Ivanovic and M. Mahar. Using Java in Embedded Systems. *Circuit Cellular Ink – The Computer Application Journal*, Issue 102, January 1999.
7. W. Kastner and C. Krügel. A New Approach for Java in Embedded Networks. In *Proceedings of the IEEE International Workshop on Factory Communication Systems*, Porto, Portugal, 2000.
8. P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, volume 22(12) of *Sigplan Notices*, pages 147–155, Orlando, FA, 1987.

9. H. Ogawa, K. Shimura, S. Matsuoka, F. Maruyama, F. Sohda, and Y. Kimura. OpenJIT Frontend System: an implementation of the reflective JIT compiler frontend. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *Lecture Notes in Computer Science*, pages 117–133. Springer-Verlag, June 2000.
10. R. J. Stroud and Z. Wu. Using Metaobject Protocols to Satisfy Non-Functional Requirements. In C. Zimmermann, editor, *Advances in Object-Oriented Metalevel Architectures and Reflection*. CRC Press, 1996.
11. Sun Microelectronics, <http://java.sun.com/j2me>. *Java 2 Platform, Micro Edition*.
12. Sun Microelectronics, <http://www.sun.com/microelectronics/picoJava/>. *picoJavaTM Microprocessor Core Overview*.
13. Sun Microelectronics, <http://java.sun.com/docs/books/tutorial/native1.1/>. *Trail: Java Native Interface*.
14. E. Tramontana. Managing Evolution Using Cooperative Designs and a Reflective Architecture. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *Lecture Notes in Computer Science*, pages 59–78. Springer-Verlag, June 2000.
15. I. Welch and R. J. Stroud. Kava – A Reflective Java Based on Bytecode Rewriting. In W. Cazzola, R. J. Stroud, and F. Tisato, editors, *Reflection and Software Engineering*, volume 1826 of *Lecture Notes in Computer Science*, pages 155–167. Springer-Verlag, June 2000.

Issues in the Design and Implementation of Efficient Interfaces between Hard and Soft Real-Time Java Components

Kelvin Nilsen and Andrew Klein

Aonix, NA, 877 S. Alvernon Way, Suite 100, Tucson, AZ 85711
{kelvin, klein}@aonix.com

Abstract. Almost all modern computer systems involve multiple layers of software, each manifesting different requirements for performance, security, and real-time predictability. Interfaces between layers are required to efficiently and reliably share information without compromising the requirements of the layers that individually participate in the interface. This paper discusses principles and empirical results in the comparison of three available techniques for interfacing between hard and soft real-time components: the Java Native Interface, the Real-Time Specification for Java, and Real-Time Core Extensions.

1 Background

For decades, principles of good software engineering have dictated strong encapsulation and partitioning of responsibility so that independently developed software components integrate cleanly into larger software systems. The implementations of modern computing environments typically include multiple layers of responsibility and trust. At the lowest layers, the operating system may itself be divided into a multi-tasking micro-kernel and interrupt handlers with the highest levels of trust, and various operating system services (e.g. caching file system, virtual memory paging, network protocol stacks) implemented as protected threads running on top of the micro-kernel. Typical application code, which runs on top of the operating system, is the least trusted. Modern high-level operating systems will protect their own integrity from violation by application code. They will also protect individual applications from being violated by each other.

Often, developers of embedded real-time systems must work at the very lowest layers of this software hierarchy. Certain hard-real-time mission-critical and safety-critical applications require that the application developer implement his own interrupt handlers and perform other operations that might compromise the integrity of the real-time operating system. Clearly, these developers must be trusted to do the right thing.

The challenges we face as providers of developer tools and real-time frameworks are to help these developers do the right thing by:

1. Making the job of developing hard-real-time components as easy and safe as possible. We do this by providing tools and reusable software components that

2. reduce the programmer's burden of responsibility and automatically verify that particular global invariants are always satisfied.
3. Providing mechanisms that enforce strong partitioning of responsibility and trust between layers of the software hierarchy. Whatever the mechanism, it must support independent realms of "trust" that are protected from one another. Any real-time development methodology that requires equal trust in all developers will never scale to the size and complexity required by today's embedded systems.
4. Assuring that the resulting technologies provide adequate efficiency in terms of memory footprint and execution speed. If a bullet-proof methodology runs so inefficiently that programmers frequently step outside its boundaries in order to get adequate performance, it is no longer bullet proof.

It needs to be emphasized that this paper focuses exclusively on issues related to the flow of information and control between the hard real-time and traditional Java environments. There are many other criteria by which available hard-real-time technologies could also be evaluated, including, for example, ease of development, portability, scalability, and maintainability of software.

1.1 Java Native Interface (JNI)

The Java Native Interface is a set of protocols that allow mixing of Java code with traditional native-compiled code written in, for example, C or C++ [1]. Using these protocols, threads started within the Java virtual machine environment can call methods implemented as native code. And operating system threads started outside the Java virtual machine environment can attach to the virtual machine and invoke methods implemented in Java.

Because of inherent differences between the Java and native programming models, the interface between these worlds is fairly expensive. For example, Java objects that are made visible to the native code must be associated with registered handles to ensure that the garbage collector does not reclaim their memory. And native code that is traversing Java data structures needs to coordinate with the garbage collector to assure data coherency between what is seen by native code, Java code, and the garbage collector itself.

In some cases, the JNI protocol makes the addresses of particular pinned Java objects visible to native code. The native code is then free to manipulate the referenced objects using traditional C-style pointer operations. When native programmers accidentally (or maliciously) reach beyond the ends of objects or overwrite certain object fields with inappropriate values, the Java virtual machine itself can be compromised. At some future time following one of these storage trampler events, the Java virtual machine may crash because its internal data structures have been corrupted.

1.2 The Real-Time Specification for Java (RTSJ)

The Real-Time Specification for Java [5] is a collection of APIs in combination with a tightening of the semantic requirements for standard Java APIs. The RTSJ requires,

for example, that traditional Java threads honor strict priority scheduling as determined by the Java thread priorities and not by operating system heuristics that occasionally age the priorities of certain threads, boosting their priorities if they have not recently run and lowering their priorities if they are seen to be consuming more than their fair share of CPU resources. The RTSJ also requires the implementation of Java's synchronization constructs to provide a means of avoiding priority inversion.

The RTSJ introduces certain new APIs that are not in standard Java. Among the extensions, RTSJ introduces two new thread types: **RealtimeThread** and **NoHeapRealtimeThread**. Unlike traditional Java threads, these threads respond to asynchronous interruption and cancellation requests. These threads are also able to run at priorities higher than the traditional virtual machine's 10 thread priorities. And the **NoHeapRealtimeThread** is guaranteed to never be preempted by garbage collection.

Within the RTSJ specification, the scheduling behavior of **RealtimeThread** is not well defined. If a particular RTSJ implementation offers some form of real-time garbage collection, then the developer can expect predictable scheduling behavior. But if it doesn't, **RealtimeThread** threads will experience unpredictable interference from garbage collection activities.

NoHeapRealtimeThread threads achieve highly predictable real-time scheduling behavior by avoiding all access to heap-allocated objects. Whenever they need access to a dynamically allocated object, they must allocate this object either from an **ImmortalMemory** or **ScopedMemory** region. **ImmortalMemory** objects live permanently and can never be reclaimed. For applications that are expected to run reliably for years at a time, the only objects that can be allocated in **ImmortalMemory** are the ones allocated during initial startup. Objects allocated within a **ScopedMemory** region are all reclaimed simultaneously, at the moment the reference count for the **ScopedMemory** region itself is decremented to zero. **ScopedMemory** regions may nest, and objects within one **ScopedMemory** region may refer to objects in outer-nested **ScopedMemory** regions, but not the other way around. Run-time checks accompany every assignment to reference fields of objects to make sure that these constraints are not violated.

Synchronization between **NoHeapRealtimeThread** and traditional Java threads is problematic given that a traditional Java thread may be preempted by garbage collection while it holds a lock on a shared object. If a **NoHeapRealtimeThread** subsequently requests access to the same synchronization lock, it may be forced to wait for garbage collection to complete before the traditional Java thread can relinquish its lock to grant access to the **NoHeapRealtimeThread**. For this reason, the RTSJ programmer is told not to synchronize between **NoHeapRealtimeThread** and traditional Java threads. Instead, any information sharing between these two domains must be realized by copying the data into **ImmortalMemory** objects and passing the copies to the other domain by way of built-in wait-free queues.

The RTSJ designers have commented in discussing the rationale for various design tradeoffs that "real time is not real fast." The primary criterion in designing the RTSJ was to enable predictable and deterministic execution. Some of the performance compromises that were implemented in order to achieve this include the requirement to impose run-time checks on reference assignments and to require copying of data

between real-time and non-real-time domains whenever information sharing is necessary.

1.3 Real-Time Core Extensions by the J Consortium (Core)

The Real-Time Core Extensions specification is similar to RTSJ in that it adds real-time programming capabilities to the standard Java virtual machine environment [6]. Both approaches partition objects so that hard-real-time threads do not depend on garbage collection activities. Insofar as is relevant to interface issues, the Core specification differs from RTSJ in the following regards:

1. The Core specification focuses only on hard-real-time applications. The intent is to serve the needs of soft-real-time developers with complementary but distinct technologies. Contrast this with the RTSJ design which promotes the use of the same APIs to serve both hard-real-time and soft-real-time requirements.
2. The Core attaches to existing Java virtual machines without requiring any changes to the existing virtual machines, or can alternatively deploy as a stand-alone execution environment. Note that RTSJ requires significant changes to the synchronization, thread scheduling, and JIT code generation models of any virtual machine it extends. To date, RTSJ extensions have only been demonstrated on J2ME.
3. The Core allows sharing of objects between the hard-real-time and traditional Java domains. All such sharing uses a carefully constructed protocol patterned after Ada's protected objects [7]. This protocol avoids the garbage collection synchronization difficulties faced by the designers of the RTSJ.
4. The Core supports neither immortal memory nor scoped memory. It offers allocation regions that are similar to immortal memory except that memory allocated in such regions can be reclaimed under programmer control. It also offers stack-allocated memory which is similar in concept to scoped memory except that the compiler enforces the restrictions required for safe operation. The design of stack allocation supports polymorphic inheritance and code reuse, guarantees absence of dangling pointers, and performs all enforcement at compile time without any requirement for run-time checks.
5. Though designers of the Core agree with RTSJ designers in observing that "real-time is not real fast", they sought to address a broader audience for whom execution speed, memory footprint, and deterministic execution were all very important. The Core is designed to offer speed, footprint, and latency comparable to C or C++ running with a modern real-time operating system.

2 Evaluation Criteria

The focus of this paper is on understanding issues related to the interface between hard-real-time and soft-real-time components. In evaluating the strengths and weaknesses of each proposed approach, we consider the issues discussed below. Since one of this paper's authors served as the editor of the J Consortium's Real-Time Core

Specification, this list of evaluation criteria closely mirrors the objectives of those who drafted that specification. We are not intentionally overlooking alternative criteria which might favor alternative interface designs. Rather, we are not fully aware of the objectives that motivated the designs of the RTSJ and of the JNI. We invite open discussion on these topics so that the general-purpose infrastructure technologies that we are developing will have broadest possible appeal and so that developers who might need to select between alternative infrastructure solutions can do so based in part on distinctions in the objectives that each infrastructure approach attempts to satisfy.

Software Engineering Ideals. Software engineering principles are intended to reduce development costs, improve software reliability, and increase generality (in order to improve reuse). The following guidelines capture our sentiment with respect to these concerns. In each case, the benefits of each objective are assumed to be independent of all others. We make no attempt to quantify the importance of individual objectives, and recognize that different situations will place different weights on the relative importance of each objective.

1. An interface design that allows the developer to think about fewer details is preferred over one that requires the developer to address more details.
2. An interface design that allows the developer to write less code is preferred over one that requires the developer to write more code.
3. An interface design that encourages composability and object-oriented inheritance is preferred over one that does not.

Certification Issues. Often, layered software systems are partitioned so as to protect more secure components from less secure components. We have commented already that there is a tendency to “trust” the developers of the lowest layer software and to put protection mechanisms in place to make sure that the lower-layer software is not compromised by the less trusted higher-level software. In order to earn the trust required of the software comprising the lowest layers of the system hierarchy, developers of that lowest-layer software usually adopt development processes that are much more rigorous than processes typical of higher level software. These processes include enforcement of rigid code style guidelines, code inspections by peers, and extensive testing driven by code coverage analysis. In some cases, such as when this software controls the flight of a commercial aircraft, the software processes include certification by an external auditing agency. Whether code is externally or internally certified, the guiding principles remain the same. Here we identify several additional criteria by which interface techniques are compared.

4. An interface that protects the low-level software from being compromised by high-level software is preferred over one that offers no such protection.
5. An interface that reduces interference by low-level software into issues pertaining to high-level software is preferred over one that does nothing to reduce such interference.

Performance. For many applications, performance is a critical development objective. This is one of the reasons that the C language is used by over 70% of embedded developers even though the language is over 30 years old and clearly does not scale well to the large embedded projects typical of today’s market. Thus, we identify two more evaluation criteria:

- 6. An interface technique that imposes very little, if any, overhead at the point that information passes through the interface is preferred over an interface technique that imposes a high overhead.
- 7. An interface technique that imposes very little, if any, overhead on the execution of typical code running in any of the software layers connected by the interface is preferred over an interface technique that does impose an overhead on execution of typical code within the independent software layers.

3 Analytical Evaluation

Each of the three alternatives for interface implementation is evaluated here according to the criteria discussed in Section . Table 1 reports our analysis of the scores, using the encodings **Good**, **Fair**, and **Poor**. The remainder of this section discusses the rationale for the assigned ratings.

Table 1. Subjective Scores for Available Interface Techniques

Evaluation Criterion	1	2	3	4	5	6	7
JNI	Poor	Poor	Poor	Good	Poor	Poor	Fair
RTSJ	Fair	Fair	Good	Fair	Good	Fair	Poor
Core	Good	Good	Good	Good	Good	Good	Good

3.1 JNI

We rate JNI poorly for all three of the software engineering criteria because JNI developers have to think very carefully about many issues when connecting native code with Java code. For example, the JNI programmer must understand non-portable details regarding the priority mappings between Java threads and native threads and must understand the relationship between the implementation of Java synchronization and the native RTOS synchronization techniques. The JNI programmer needs to carefully design techniques to avoid priority inversion. Accessing Java objects from C code requires strict adherence to special protocols designed to abstract Java object representations. This requires much more code than would be required if the C program were dealing with traditional C structures. Finally, the JNI interface does not

enforce any type checking or encapsulation and does not support any sort of object-oriented inheritance.

We rate JNI as good in category 4 because Java's security model is very good at protecting native components from being compromised by Java application software. On the other hand, we rate JNI as poor in category 5 because there are no protections in place to prevent native code from compromising the integrity of the Java virtual machine.

We rate the interface efficiency as poor because each time control passes between native and Java code, significant marshalling of information must take place. The typical JNI call is more than ten times as expensive as a typical Java method invocation.

In general, it would appear that the performance of traditional C code and traditional Java code are unaffected by the presence of the JNI interface. Nevertheless, we rate this performance as mediocre for two reasons. First, when C code is accessing objects shared with the Java virtual machine, it runs much more slowly than when it is accessing traditional C structures residing in its own heap. This is because the C code must follow special protocols involving macros and function invocations to access the Java objects. Second, the requirement to efficiently support the JNI interface imposes implementation tradeoffs on the Java virtual machine architecture which slows execution of traditional Java code even if that code is not making use of the JNI interface.

3.2 RTSJ

RTSJ represents an improvement over JNI in the details that must be considered and in the code that must be written to implement both sides of the interface. Thus, we rate RTSJ as fair for both criteria 1 and 2. One reason we do not offer good ratings is because **NoHeapRealtimeThreads** cannot use traditional Java techniques to synchronize with traditional **java.lang.Threads** or with **RealTimeThreads**. Instead, they must devise ad hoc sharing protocols that allow certain objects to exist at some times within the domain of the hard-real-time world and at other times within the domain of the soft-real-time or non-real-time world. RTSJ allows objects to be passed between domains by placing them onto specially designed wait-free queues. The only objects that can be shared between these two worlds are objects residing in **ImmortalMemory**. Since these objects are not subject to automatic garbage collection, developers must take responsibility for manually managing the allocation and recycling of such objects as they move in and out of the hard-real-time domain. Most developers of hard real-time code program defensively. They will generally take extra precautions to protect the integrity of their hard real-time software from interference by other software components. Because of this, most will be wary of releasing their critical data structures to the control of non-real-time software components. Instead, they will often desire to keep their critical data structures to themselves and will only copy information into shared objects when necessary to communicate with non-real-time components. Finally, note that RTSJ programmers must invent their own ad hoc techniques for throttling the flow of information through a wait-free queue. What is the real-time program to do if it cannot place critical information into the queue because the queue is full, or if a reusable buffer does not

come back from the non-real-time world in sufficient time to fill it with the next requested information?

We rate RTSJ as good in category 3 because information is passed between hard-real-time and non-real-time domains in the form of Java objects. We emphasize that these are special objects (residing in **ImmortalMemory**) and special care must be taken to ensure that the methods associated with these objects can be invoked both from within a **NoHeapRealtimeThread** and a **java.lang.Thread**. We do not penalize RTSJ in the category 3 rating for these issues because we consider RTSJ to have already paid the price of these sins in its category 1 and 2 ratings.

We rate RTSJ as mediocre in protecting the hard-real-time domain from mistakes made within the traditional Java domain. On the one hand, RTSJ uses standard Java encapsulation techniques to protect hard-real-time components from traditional components. On the other hand, there is no other partitioning beyond this to protect the hard-real-time domain from the Java domain. The two domains are so tightly integrated that if the Java virtual machine crashes, the hard-real-time environment also crashes. If a traditional Java thread consumes all of the virtual machine's memory, the hard-real-time environment will not be able to find memory to represent scoped memory regions. Also, if the traditional Java code fails to return the immortal objects that carry information by way of wait-free queues from the hard-real-time domain to the Java domain, the hard-real-time application software will not be able to continue to send information to the traditional Java domain.

With the RTSJ, **NoHeapRealtimeThreads** are not allowed to access traditional heap objects. This helps to prevent **NoHeapRealtimeThreads** from interfering with the activities of the traditional Java application code. Thus we give category 5 a good rating. Unfortunately, most real-time developers would rather have a higher rating in category 4 than in category 5.

We rate category 6 as fair because the protocols for sharing information between the hard-real-time and non-real-time domain are much heavier than traditional Java-synchronized sharing. Passing information between domains usually requires allocation of a reusable buffer from an existing pool and copying of information into the reusable buffer. Then the object must be inserted onto one end of a wait-free queue and subsequently removed from the other end of the wait-free queue. Even so, this interface is more efficient than the typical JNI implementation.

We rate category 7 as poor because the RTSJ imposes a number of run-time checks on code executing both within the traditional Java domain and within the hard-real-time domain. These checks are required to enforce that **NoHeapRealtimeThreads** never depend on or interfere with operation of the garbage collector.

3.3 Core

We assign good ratings for all three of the software engineering criteria. Using the Real-Time Core Extensions, the interface between hard-real-time and traditional Java objects is represented by a core-baseline method. To the traditional Java developer, this looks like a standard object-oriented Java method invocation. The hard-real-time developer who implements the core-baseline method writes this code the same way he would write all of the other code in his hard-real-time application. He inserts a special

marker into the body of the method to inform the core compiler that it must translate this method according to the calling conventions of the traditional Java execution environment rather than the streamlined conventions of the hard-real-time environment. All of the implementation complexity is hidden behind the core compiler. Furthermore, because the interface is implemented using standard Java object-oriented disciplines, it takes full advantage of inheritance and polymorphism.

Categories 4 and 5 also earn good scores. There is a strong partition between the hard-real-time and traditional Java domains, so strong that the hard-real-time execution environment can continue to run even if the Java virtual machine crashes. Memory is permanently partitioned between the two domains so there is no risk that a runaway Java application will consume memory that ought to be reserved for hard-real-time activities. Further isolation is provided by compile-time enforcement that prevents traditional Java objects from directly manipulating core objects and prevents core objects from referring to traditional Java objects. A protocol is provided to allow traditional Java objects to obtain handles to specific core objects, but these handles can only be used to execute the core-baseline methods that were implemented by the trusted hard-real-time developer. There is no way for a traditional Java thread to access the instance variables of the core objects.

The strong partitioning that exists between the hard-real-time and traditional Java domains enables a very efficient interface implementation. Because the core domain never refers to traditional Java objects, there is no need to marshal object representations on this interface. And even though we prohibit traditional Java threads from directly accessing the instance variables of core objects, the core programmer can write very efficient accessor methods which will be compiled and even in-lined by the core compiler. This particular protocol was designed to allow traditional Java threads to run according to their standard code model, without any changes required to the Java virtual machine when it is attached to a core execution environment. Likewise, the protocol was designed to allow the hard-real-time Java threads to run at full speed comparable to C or C++ code. No run-time checks are required. Thus we give good ratings to the Core for categories 6 and 7 as well.

4 Empirical Evaluation

Aonix is currently implementing a subset of the Core specification. In this section, we provide a summary of empirical results obtained by measuring the alternative interface technologies as they .

4.1 Experimental Benchmarks

Two representative applications were selected for the purpose of evaluating the interface between hard-real-time and soft-real-time components. These are synthetic benchmarks based on real-world scenarios. We describe the two benchmarks below. Additional representative workloads are under development. Source code for the two benchmark applications described below and for the additional benchmark applications that are under development will be available from the authors of this paper.

Zero-Copy Network Stack. This application represents a simplification of the sort of work carried out by a network router or gateway.

The hard real-time component represents the lower level control plane of a network protocol stack. This routine repeatedly selects a random packet from a collection of 128 previously allocated and initialized packets. It examines certain fields of the packet to determine if this packet requires any special handling. In our simulation, an average of one packet in every 17 requires special handling and is forwarded to the soft-real-time management-plane software for further processing. The packets that do not need any special handling are queued directly for another hard-real-time thread to do further output processing.

The soft-real-time management plane examines each packet that is forwarded for special handling, makes some minor adjustments to the content of the packet, and then queues the packet for further output processing within the hard-real-time domain.

For our benchmark, we measure how much time is required to process 1,000 packets through the soft-real-time code. This corresponds to total processing of approximately 17,000 packets. Although our benchmark result is reported as an elapsed time, this value is inversely proportional to the maximum rate at which packets could be reliably handled by a network element implemented based on the respective interface technology.

Streaming Data Recorder. This application represents the sort of work that might be carried out in a streaming data recorder that is part of, for example, a seismic monitor or a black-box recorder for a commercial aircraft.

In this application, the hard-real-time component is responsible for writing buffered data to the physical non-volatile media. The soft-real-time component is responsible for gathering data samples to be recorded. In this simulation, the soft-real-time component simply generates random integer values and stores them into a buffer holding 256 entries. Each time the buffer fills, the soft-real-time component hands the buffer over to the hard-real-time domain and begins filling a second buffer. The hard-real-time component simulates the streaming data write operation by summing the integers in the buffer and then sleeping for 100 microseconds before returning the emptied buffer back to the soft-real-time component.

For this benchmark, we measure how much time is required to process 16,384 buffers of 256 entries each. As with the other benchmark, this result is also reported as an elapsed time. The reported value is inversely proportional to the maximum rate at which information could be collected and streamed to non-volatile media using the respective interface technology.

Comparison Between Workloads. The zero-copy network stack is very typical of layered network protocol implementations in which 90% of the CPU time is spent in 10% of the code. As is typical in such systems, the 10% of the code that is most performance critical is the lower level hard-real-time code. In this example, only 1 out of 17 packets is bubbled up to the soft-real-time management layer. And even when a packet is sent to the management layer for special processing, the management layer only examines and manipulates selected fields of the packet's header. In contrast, the lower layer software processes every packet and deals with the entire data payload of each packet. This benchmark places greatest emphasis on performance of the low-level hard-real-time code. The need to pass information from the hard-real-time to

soft-real-time world is relatively rare, as is the need to examine or manipulate shared objects from the soft-real-time domain.

The streaming data recorder provides a much more balanced sharing of computation between the soft-real-time and hard-real-time domains. For each entry in the buffer, the soft-real-time thread generates a random number and inserts this into the buffer. Similarly, the hard-real-time component examines each entry in the buffer as a simulated step in the streaming write operation. Thus, this benchmark places approximately equal emphasis on efficiency of soft-real-time and hard-real-time access to shared objects. The shared buffer objects themselves are passed between the hard-real-time and soft-real-time domains relatively rarely in comparison to the frequency of operations that access the fields of the shared objects.

4.2 Measured Platforms

We implemented the two sample applications using each of the three interface techniques. We ran all of our measurements on the same computer, a dual-processor Pentium II processor running at 350 MHz. The system has 256 MBytes of RAM. It is running Timesys Linux version 3.2.214smp, configured to use only one of the processors. We chose to run in single-processor configuration so as to reduce uncertainty as to what exactly is being measured. Except for the measured workload, each measurement was taken with the processor in a quiescent state.

JNI. To measure the cost of the JNI interface, we ran the JNI versions of the two applications on a pre-release version of PERC 4.1. All shared data objects were allocated in the Java heap. We chose to measure JNI performance on PERC rather than Sun's JDK product because the Java threads are to represent soft-real-time behavior.

RTSJ. We evaluated the efficiency of the RTSJ interface by measuring the RTSJ version of the two applications running on RTSJ reference implementation 1.0. For this test workload, all of the shared objects were allocated from within **ImmutableMemory**. Hard-real-time activities were represented by **NoHeapRealtimeThread** objects. We chose to represent soft-real-time activities with **java.lang.Thread** objects. This represents a compromise from the original intent of the demonstration, forced by the lack of real-time garbage collection in existing RTSJ implementations.

The hard-real-time components trusted the soft-real-time Java code to deal appropriately with the packet and buffer objects that were shared between the two domains. Thus, it was not considered necessary to copy data out of private hard-real-time buffers before communicating their contents to firm-real-time Java components.

We are aware that TimeSys has now completed a commercial implementation of the RTSJ, named JTime, and we would have preferred to evaluate the efficiency of the RTSJ interface by measuring that implementation. Unfortunately, Timesys refused our repeated requests to license the JTime implementation for the purposes of conducting this research. Individuals interested in the performance of JTime may obtain test programs directly from the authors and conduct their own tests of the JTime platform.

Core. We have not yet completed implementation of the Real-Time Core Extensions so the measurements reported here are only approximations based on the currently available partial implementation. In the Core programs, all shared objects are allocated from within the core domain. In the simulation, the core components are represented by C code written to follow the sharing protocol that was designed for integration of core components. We consider this a fair representation of core performance, especially in light of the fact that one of our next implementation steps is to automate the translation of real-time Core code into the equivalent C code.

For purposes of this simulation, we have instrumented a version of PERC 4.0 to recognize classes that implement a particular interface and were loaded by a particular class loader as having core-baseline methods. These methods are treated like JNI methods but with a less costly (more efficient) interface implementation.

One notable weakness of the existing partial implementation is that all access to shared Core objects from within the Java domain must include an invocation of a native method. A future version of the Core implementation will in-line the implementation of many core-baseline methods.

4.3 Experimental Results

The results of our empirical evaluation are tabulated in Table 2.

Table 2. Benchmark measurements (ms)

Benchmark		Trial 1	Trial 2	Trial 3	Average
Zero-Copy Network Stack	JNI	27,944	21,315	23,159	24,139
	RTSJ	10,395	10,359	10,309	10,354
	Core	2,160	2,090	2,060	2,103
Streaming Data Recorder	JNI	4,802	4,757	4,749	4,769
	RTSJ	191,694	191,563	191,516	191,591
	Core	8,920	9,000	8,850	8,923

Discussion of Results. From the first benchmark, it is interesting to note that even though C code is known generally to be much faster than Java code, the JNI implementation runs over 10 times slower than the Core solution and over twice as slow as the RTSJ approach. This is presumably because of the very high cost of “crossing the barrier” between the JNI and JVM domains. The Core hard-real-time components are also written in C, but the protocol for sharing objects between the

Core and JVM domains is much simpler. This Core sharing protocol is independent of JVM technologies so the benefits shown here would accrue to integration with other JVMs as well.

In the second benchmark, the costs of crossing the interface between hard- and soft-real-time domains are less significant. Much more significant is the efficiency with which each approach independently executes soft-real-time and hard-real-time components. RTSJ is over 20 times slower than Core. This is presumably because all of the RTSJ components are interpreted, whereas the Core solution benefits from the PERC JIT compiler. Note that interpretation of RTSJ code runs even slower than interpretation of traditional Java code because RTSJ requires extra run-time checks to enforce memory partitioning and dynamic scoping.

Presumably, the commercial JTime implementation would perform better than the RTSJ reference implementation. However, we were not given an opportunity to evaluate that environment. It is our understanding that performance even of JTime, which offers limited forms of ahead-of-time compilation, is generally disappointing, and this may be one of the reasons Timesys has elected to protect the product from public scrutiny. Our difficulties obtaining access to the commercial product hint of larger issues that must be addressed in the real-time marketplace. Discussion of economic issues that accompany dependency on proprietary technical products focused on narrow market niches are beyond the scope of this paper.

Given the huge penalty paid by JNI in the first benchmark, it is surprising to see it perform significantly better than the other two alternatives for this benchmark. Our analysis of this anomaly is that this particular benchmark was particularly sensitive to a shortcoming in our prototype Core implementation. In particular, for each value inserted into the buffer, the soft-real-time component must make a core-native method call. This is because the data buffer is a core object which does not live in the Java heap and does not follow the traditional Java heap-access protocols. Even though the protocol for core-native method calls is much more efficient than the protocol for traditional JNI native method calls, it is still a fairly expensive operation. A future improvement to our Core implementation will replace the core-native method calls with in-lined machine code. This in-lined machine code will run even faster than JIT-translated Java code because it will circumvent the protocols required for coordination with garbage collection and relocation of objects. Thus, we are confident that a future version of the Core will run significantly faster than the JNI implementation on the Streaming Data Recorder application.

There is one last point we desire to emphasize relating to these two benchmarks. It is important to note that the alternative technical approaches yield radically different performance, spanning a range of over 40-fold. Even more noteworthy, the best technology for one of these challenges was the worst for the other! These observations underscore the importance of (a) recognizing that different technologies are better suited to different problems, (b) it is very difficult to predict the performance of a given technology on your specific problem based on the experience of others with other problems, and finally (c) the stakes are quite high as are the risks (we're not talking about performance variations of only plus or minus 30%; we're talking about orders of magnitude). This work is ongoing. Aonix will continue to expand its repertoire of test applications in order to better understand the tradeoffs represented by particular interface technologies.

Qualitative Observations. This was the authors' first experience developing software using the RTSJ APIs. Briefly, we summarize here some of the lessons we learned from this experience.

First, we found it was easier than originally anticipated to share objects between the hard- and soft-real-time domains. However, there are subtle coordination issues, the importance of which should not be underestimated, and we expect that less "formal" developers would be likely to stumble into certain pitfalls, mentioned below. We used wait-free queues to pass objects between the two domains. We were confused by the documentation which states, for example, that the `read()` method of a `WaitFreeWriteQueue` may be called by more than one writer [*sic*], which seems to contradict the constructor API, which requires as one of its arguments a reference to the `java.lang.Thread` object that represents the writer. For our application, we needed only one writer, so this was not an issue for us. But we were left wondering how different RTSJ implementors might end up interpreting the specification in different ways, leading to incompatibilities between code developed for one implementation versus another.

For both of our applications, we would have liked to have had a built-in blocking queue upon which a soft-real-time thread could block waiting for information from the hard-real-time domain, and upon which a hard-real-time thread could block waiting for a slot in the queue to become available for transmission of information to the soft-real-time domain. In the absence of this, we were forced to introduce busy loops that would repeatedly poll the queue status and sleep until the queue was ready. We were tempted to implement our own blocking queue. All of the facilities are in place to make the implementation quite straightforward. This, however, is where an unwary RTSJ programmer may create problems for himself, or even worse, for the folks down the hall who are writing code that has to run on the same platform. The problem is as follows:

1. Suppose you have a `NoHeapRealtimeThread` running at real-time priority 20, and you decide that you don't mind if this particular thread blocks waiting for certain coordination with a non-real-time thread. So you use a traditional Java monitor to arrange that the `NoHeapRealtimeThread` will block until the Java thread performs a certain action.
2. Suppose further that the `NoHeapRealtimeThread` attempts to enter this shared monitor in order to see if the Java thread is ready for a coordinated handoff, but the monitor is locked because the Java thread is already accessing certain shared data structures associated with this monitor.
3. Because of the way the RTSJ specification is written, the `NoHeapRealtimeThread` will endow its priority to the Java thread, allowing it to run at real-time priority 20 until it advances to the point of releasing its lock on the shared monitor.
4. And finally, suppose the Java thread cannot make forward progress because the Java VM is in the middle of garbage collection. The Java thread, and the garbage collector upon which it depends for its forward progress, are now running at effective priority 20 and will continue to do so until garbage collection completes.

The programmer who introduced this shared monitor thought it would be ok because, as far as he was concerned, his real-time task had nothing to do until the Java domain produced another value for him to process. The problem is that every other

NoHeapRealtimeThread with priority equal or lower than 20 is also prevented from running while garbage collection completes. In our opinion, it's a little too easy for programmers to introduce these sorts of priority inversion bugs into an RTSJ system. These sorts of problems are even more troubling than the scoped-memory incompatibilities that RTSJ developers encounter when they attempt to invoke existing Java library code from within **NoHeapRealtimeThreads**. When library code invoked from a **NoHeapRealtimeThread** violates the scoped-memory restrictions, this throws an exception, so the programmer can figure out that these libraries are incompatible with **NoHeapRealtimeThreads**. No such exception is thrown when library code attempts to synchronize on objects that might be shared between the hard-real-time and non-real-time domains.

5 Conclusions

There are various ways to connect soft-real-time and hard-real-time software components together. Each has particular strengths and weaknesses. When evaluated according to the criteria detailed in this paper, the Real-Time Core approach is superior to available alternatives. Further research may reveal alternative criteria that are of equal or greater importance to particular problem spaces and may reveal additional ways to evaluate alternative approaches in terms of the criteria already identified.

Acknowledgments. We thank the paper's reviewers for several insightful comments. We also acknowledge support from the U.S. Navy under contract N00178-08-C-3087.

References

1. Liang, S. "The Java Native Interface Programmer's Guide and Specification", Addison-Wesley Publishing Co. 320 pages. June 10, 1999.
2. Zukowski, J. "Mastering Java 2, J2SE 1.4", Sybex. 928 pages. April 2002.
3. Keogh, J. "J2EE: The Complete Reference", McGraw-Hill Osborne Media. 904 pages. Sept. 6, 2002.
4. Keogh, J. "J2ME: The Complete Reference", McGraw-Hill Osborne Media. 768 pages. Feb. 27, 2003.
5. Bollella, G. et al. "The Real-Time Specification for Java", Addison-Wesley Publishing Company. 195 pages. Jan. 15, 2000.
6. "Real-Time Core Extensions", J Consortium. 170 pages. Sept. 2, 2000.
7. Barnes, J. "Programming in Ada 95". Addison-Wesley Publishing Company. 720 pages. June 10, 1998.

The Metronome: A Simpler Approach to Garbage Collection in Real-Time Systems

David F. Bacon, Perry Cheng, and V.T. Rajan

IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.
dfb@watson.ibm.com
{perryche, vtrajan}@us.ibm.com

Abstract. With the wide-spread adoption of Java, there is significant interest in using the language for programming real-time systems. The community has generally viewed a truly real-time garbage collector as being impossible to build, and has instead focused its efforts on adding manual memory management mechanisms to Java. Unfortunately, these mechanisms are an awkward fit for the language: they introduce significant run-time overhead, introduce run-time memory access exceptions, and greatly complicate the development of library code. In recent work we have shown that it is possible to build a real-time collector for Java with highly regular CPU utilization and greatly reduced memory footprint. The system currently achieves 6 ms pause times with 50% CPU utilization (MMU) and virtually no “tail” in the distribution. We show how this work can be incorporated into a general real-time framework, and extended to systems with higher task frequencies. We argue that the community should focus more effort on such a simple, orthogonal solution that is true to the spirit of the Java language.

1 Introduction

Garbage collected languages like Java are making significant inroads into domains with hard real-time concerns, such as automotive command-and-control systems. However, the engineering and product life-cycle advantages consequent from the simplicity of programming with garbage collection remain unavailable for use in the core functionality of such systems, where hard real-time constraints must be met. As a result, real-time programming requires the use of multiple languages, or at least (in the case of the Real-Time Specification for Java [4], or RTSJ) two programming models within the same language. Therefore, there is a pressing practical need for a system that can provide real-time guarantees for Java without imposing major penalties in space or time.

In previous work [2,1], we presented the design and evaluation of a uniprocessor collector that is able to achieve high CPU utilization during collection with far less memory overhead than previous real-time garbage collectors, and that is able to guarantee time and space bounds provided that the application can be accurately characterized in terms of its maximum live memory and average allocation rate over a collection interval.

In this position paper, we begin by describing the weakness of the programming model introduced by the RTSJ, both in terms of usability by the programmer and in terms of burdens it places on the virtual machine.

We then provide a brief overview of the features of our collector, describe how it can be applied to create a far simpler real-time programming interface, and discuss how to improve its resolution so that it can be used to program systems that require response times in the tens of microseconds.

2 Problems with RTSJ

The Real-Time Specification for Java (RTSJ) is a standard for extending Java to meet the needs of real-time applications. The specification identifies seven areas of interest. The two areas relevant to this paper are thread scheduling and memory management. The design of RTSJ memory management is heavily influenced by a desire to “allow the allocation and reclamation of objects outside of any interference by any GC algorithm”. This policy arose from the expert group’s belief that garbage collection alone could not sufficiently meet real-time needs. Instead, real-time threads (**NoHeapRealTimeThreads**) can allocate and manipulate objects only from two new memory areas (the immortal heap and manually programmed scoped memory regions) which are free from GC interference. The (regular) heap is still managed by the garbage collection. To maintain the separation between these regimes, there are restrictions on references between objects from different memory regions. The goal is to enable the collection of scoped memory regions independent of the heap and for the real-time threads to always be able to pre-empt the GC thread.

In this section, we will examine the aspects of the RTSJ design that relate to memory management and evaluate the design’s effectiveness in providing an overall real-time solution to the problem of memory management.

2.1 Description

RTSJ provides two additional types of real-time threads. Running at the lowest priority are traditional Java threads which are subject to the pauses introduced by the garbage collector. RTSJ’s **RealTimeThread** run at a higher priority than the garbage collector but because it can access the heap, cannot arbitrarily pre-empt the GC. Instead it must be delayed for up to the GC-induced latency. The standard does not require this latency to be low. In the case that a stop-the-world collector is used, a **RealTimeThread** is no better than a regular thread. Finally, RTSJ’s **NoHeapRealTimeThread** can pre-empt the GC at any moment and any encountered latency is due to the cost of context switching and scheduler computations. To support this, it is illegal for such threads to manipulate or refer to any object in the heap.

The severe restrictions on **NoHeapRealTimeThread** is ameliorated by the introduction of additional memory areas. We focus on the immortal heap and scoped memory regions. Objects allocated into the immortal heap have lifetimes for the remaining duration of the program. It is possible for objects allocated in the immortal heap to become inaccessible but they remain alive simply by virtue of residing in this region. Secondly, scoped memory regions support objects whose lifetimes are shorter in duration than the duration of the entire application and follow a LIFO pattern. A scoped memory region supports memory allocation but does not necessarily support garbage collection. The

scoped memory's size is chosen at its creation time. The set of all scoped memory regions form a tree with the immortal region and the heap as the implicit root. This graph is dynamically implied by the stack-like order in which different threads enter and exit these scoped regions. A thread will, by default, allocate objects in the scope it is currently in. Objects residing in scoped regions can only refer to objects residing in an outer scope (i.e., an ancestral scope). Since the immortal region and the heap form the root node, objects in a scope can refer to an object in the immortal region or the heap, but not vice versa. The overall effect of the scopes is that once all threads leave a scope, the entire scoped memory region can be recycled without tracing through the region.

2.2 Barriers

To enforce the pointer restrictions above, RTSJ uses runtime checks that will throw either an `IllegalAssignmentError` or a `MemoryAccessError` if an operation is about to violate the conditions. Specifically, whenever a reference is about to be loaded, a *read barrier* will throw the `MemoryAccessError` exception if the executing thread is a `NoHeapRealtimeThread` and the loaded value resides in the heap. Secondly, whenever object X is being stored into object Y, a *write barrier* will throw the `IllegalAssignmentError` if the scope of X is not an outer scope of the scope of Y [7].

This particular read barrier is hard to optimize. First, coalescing nearby read barriers on the same object is difficult because the barrier is field-dependent. That is, if different fields of an object X are accessed, the read barrier must be applied repeatedly to each access since the exception is dependent not on where X resides but on the contents of the fields. Second, since a method can be executed by both regular threads and `NoHeapRealTimeThreads`, the read barrier must check the thread type at each barrier. Even coalescing thread checks within a single method is difficult without affecting the other part of barrier.

The write barrier suffers from two factors. Even if the source and target objects of the assignment are not in scoped regions, a dynamic test to ensure this is required. In cases where the objects are in scoped areas, the test must first determine the object's scopes and then determine whether one scope is an outer scope of another scope. The overall write barrier will likely include several memory operations and several conditional instructions.

It is worth emphasizing that these barriers are imposed by the RTSJ and are entirely separate from the barriers, if any, that a particular garbage collector might impose.

2.3 Difficult Usage

In RTSJ, only the `NoHeapRealTimeThread` is guaranteed true pre-emption. However, it is unclear how to program such a thread if it needs to allocate objects whose lifetimes are unclear. That is, objects that are neither immortal nor follow the LIFO pattern of scoped regions. However, RTSJ does provide wait-free queues that allow real-time thread to safely synchronize with other threads without priority inversion problems.

Consider a real-time server where the high-priority `NoHeapRealTimeThread` handles incoming queries and sends the request to a lower priority regular thread via a wait-free queue. Assuming that the request is a `String` object, choosing where to allocate the

String object is problematic. Clearly the object is not immortal and allocating it from the immortal region will lead to a memory leak if the server is up for any appreciable time. If one were to allocate the object in a scoped region, the leakage problem becomes when the scope would be exited because it is possible that the scope would always contain at least the most recent request. It is probably possible to overcome this scenario by appropriate synchronization on when to enter and exit scopes along with data-copying. However, it seems clear there are programming patterns (for example, FIFO) that are inexpressible or hard to express with scoped memory regions.

Aside from the problem of expressability, there is also the pragmatic problem of existing library code. The vast majority of library code does not use RTSJ features and therefore allocate objects in the regular heap. As a result, any `NoHeapRealTimeThread` that uses the library code will result in an exception being thrown at run-time.

2.4 Fragmentation

The separation of memory into scoped regions also burdens the programmer with determining the maximum size of the each scoped memory area necessary for execution. In contrast, a regular Java program needs to determine only a single parameter for the heap size. The need to determine memory usage in such a fine-grained manner may require over-provisioning. Consider a thread that enters an outer scope A and an inner scope B and will allocate a total of 100 KB among the two scopes. If the distribution of the 100 objects is unknown until the scopes are entered, then both scopes must be able to accommodate 100 KB, resulting in a total memory usage of 200 KB. On the other hand, allocating the objects in the heap would require only 100 KB. This overhead is above and beyond the wastage associated with not garbage collecting a region.

3 Overview of the Metronome

We begin by summarizing the results of our previous work [2,1] and describing the algorithm and engineering of the collector in sufficient detail to serve as a basis for understanding the work described in this paper.

Our collector, the Metronome, is an incremental uni-processor collector targeted at embedded systems. It uses a hybrid approach of non-copying mark-sweep (in the common case) and copying collection (when fragmentation occurs).

The collector is a snapshot-at-the-beginning algorithm that allocates objects black (marked). While it has been argued that such a collector can increase floating garbage, the worst-case performance is no different from other approaches and the termination condition is easier to enforce. Other real-time collectors have used a similar approach.

Figures 1 and 2 show the real-time performance of our collector. Unlike previous real-time collectors, there is no “tail” in the distribution of pause times, CPU utilization remains very close to the target, and memory overhead is low — comparable to the requirements of stop-the-world collectors. In this section we explain how the Metronome achieves these goals.

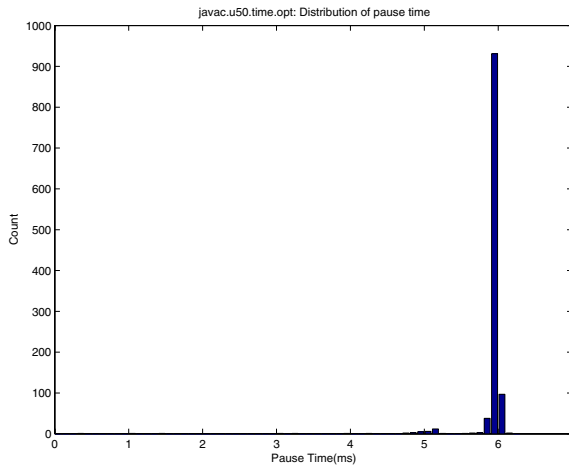


Fig. 1. Pause time distributions for javac in the Metronome, with target maximum pause time of 6 ms. Note the absence of a “tail” after the target time.

3.1 Features of Our Collector

Our collector is based on the following principles:

Segregated Free Lists. Allocation is performed using segregated free lists. Memory is divided into fixed-sized pages, and each page is divided into blocks of a particular size. Objects are allocated from the smallest size class that can contain the object.

Mostly Non-copying. Since fragmentation is rare, objects are usually not moved.

Defragmentation. If a page becomes fragmented due to garbage collection, its objects are moved to another (mostly full) page.

Read Barrier. Relocation of objects is achieved by using a forwarding pointer located in the header of each object [5]. A read barrier maintains a to-space invariant (mutators always see objects in the to-space).

Incremental Mark-Sweep. Collection is a standard incremental mark-sweep similar to Yuasa’s snapshot-at-the-beginning algorithm [8] implemented with a weak tri-color invariant. We extend traversal during marking so that it redirects any pointers pointing at from-space so they point at to-space. Therefore, at the end of a marking phase, the relocated objects of the previous collection can be freed.

Arraylets. Large arrays are broken into fixed-size pieces (which we call arraylets) to bound the work of scanning or copying an array and to bound external fragmentation caused by large objects.

Since our collector is not concurrent, we explicitly control the interleaving of the mutator and the collector. We use the term *collection* to refer to a complete mark/sweep/defragment cycle and the term *collector quantum* to refer to a scheduler quantum in which the collector runs.

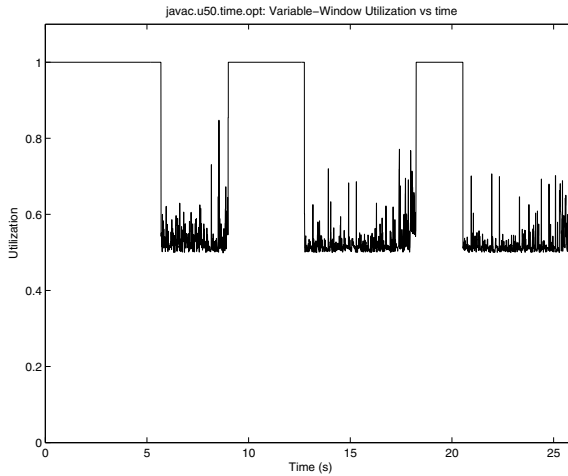


Fig. 2. CPU utilization for javac under the Metronome. Mutator interval is 6 ms, collector interval is 6 ms, for an overall utilization target of 50%; the collector achieves this within 3% variation.

3.2 Read Barrier

We use a Brooks-style read barrier [5]: each object contains a forwarding pointer that normally points to itself, but when the object has been moved, points to the moved object.

Our collector thus maintains a to-space invariant: the mutator always sees the new version of an object. However, the sets comprising from-space and to-space have a large intersection, rather than being completely disjoint as in a pure copying collector.

While we use a read barrier and a to-space invariant, our collector does not suffer from variations in mutator utilization because all of the work of finding and moving objects is performed by the collector.

Read barriers, especially when implemented in software, are frequently avoided because they are considered to be too costly. We have shown that this is not the case when they are implemented carefully in an optimizing compiler and the compiler is able to optimize the barriers.

We apply a number of optimizations to reduce the cost of read barriers, including well-known optimizations like common subexpression elimination, as well as special-purpose optimizations like barrier-sinking, in which we sink the barrier down to its point of use, which allows the null-check required by the Java object dereference to be folded into the null-check required by the barrier (since the pointer can be null, the barrier can not perform the forwarding unconditionally).

This optimization works with whatever null-checking approach is used by the run-time system, whether via explicit comparisons or implicit traps on null dereferences. The important point is that we usually avoid introducing extra explicit checks for null, and we guarantee that any exception due to a null pointer occurs at the same place as it would have in the original program.

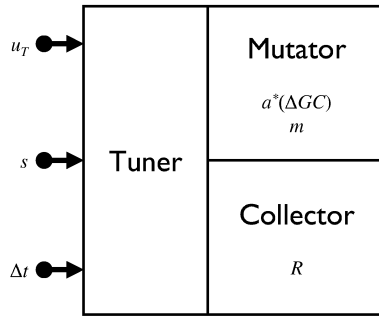


Fig. 3. Tuning the performance of an application (mutator) with the collector. The mutator and collector each have certain intrinsic properties (for the mutator, the allocation rate over the time interval of a collection, and the maximum live memory usage; for the collector, the rate at which memory can be traced). In addition, the user can select, at a given time resolution, either the utilization or the space bound (the other parameter will be dependent).

The result of our optimizations is that for the SPECjvm98 benchmarks, read barriers only have a mean cost of only 4%, or 9.6% in the worst case (in the 201.compress benchmark).

3.3 Time-Based Scheduling

Our collector can use either time- or work-based scheduling. Most previous work on real-time garbage collection, starting with Baker's algorithm [3], has used work-based scheduling. Work-based algorithms may achieve short individual pause times, but are unable to achieve consistent utilization.

The reason for this is simple: work-based algorithms do a little bit of collection work each time the mutator allocates memory. The idea is that by keeping this interruption short, the work of collection will naturally be spread evenly throughout the application. Unfortunately, programs are not uniform in their allocation behavior over short time scales; rather, they are bursty. As a result, work-based strategies suffer from very poor mutator utilization during such bursts of allocation.

In fact, we showed both analytically and experimentally that work-based collectors are subject to these problems and that utilization often drops to 0 at real-time intervals.

Time-based scheduling simply interleaves the collector and the mutator on a fixed schedule. While there has been concern that time-based systems may be subject to space explosion, we have shown that in fact they are quite stable, and only require a small number of coarse parameters that describe the application's memory characteristics in order to function within well-controlled space bounds.

3.4 Provable Real-Time Bounds

Our collector achieves guaranteed performance provided the application is correctly characterized by the user. In particular, the user must be able to specify the maximum

amount of simultaneously live data m as well as the peak allocation rate over the time interval of a garbage collection $a^*(\Delta GC)$. The collector is parameterized by its tracing rate R .

Given these characteristics of the mutator and the collector, the user then has the ability to tune the performance of the system using three inter-related parameters: total memory consumption s , minimum guaranteed CPU utilization u_T , and the resolution at which the utilization is calculated Δt .

The relationship between these parameters is shown graphically in Figure 3. The mutator is characterized by its allocation rate over the interval of a garbage collection $a^*(\Delta GC)$ and by its maximum memory requirement m . The collector is characterized by its collection rate R . The tunable parameters are Δt , the frequency at which the collector is scheduled, and either the CPU utilization level of the application u_T (in which case a memory size s is determined), or a memory size s which determines the utilization level u_T .

Note that in either case both space and time bounds are guaranteed.

4 Integrating the Metronome with a Real-Time System

As we showed in Section 2, the RTSJ treats garbage collection as a foreign entity, outside of the normal scheduling and priority mechanisms of the system. This in turn leads to the requirement to create various different types of memory regions, with complex restrictions on which regions and thread types can reference other regions. The end result is a system which lacks orthogonality, introduces unpredictable run-time exceptions, and makes development and understanding of library code extremely difficult.

We advocate a different approach: integrating collection into the run-time system, and particularly, into the scheduler, in such a way that garbage collection is a real-time task like all others.

The benefits of this approach are enormous in terms of simplification of the programming model. Since one of the major benefits of Java is its reliability and simplicity, we believe this is fundamental to the spirit of a real-time implementation of Java.

A model based on a truly real-time collector is simpler in the following ways:

- only a single memory space;
- no run-time exceptions on memory accesses;
- ability to share objects between real-time and non-real-time threads; and
- ability of real-time threads to call standard library routines.

In the previous section we presented an overview of the Metronome and how it schedules garbage collection. This scheduling approach can easily be adapted to periodic real-time scheduling. The interval Δt is the period of the collector. The utilization u_T is the fraction of that time devoted to the collector. The user parameterizes the application in terms of its allocation rate, which is already an RTSJ parameter on real-time threads. The one additional required parameter is the maximum memory utilization m .

With these parameters, the garbage collector becomes a periodic real-time task. The time remaining after garbage collection, $1 - u_T$, is the time available in which to

schedule the high-priority real-time tasks. A feasible schedule must be able to perform the real-time tasks in this interval.

Of course, the main limitation of this approach is that collection can consume a significant portion of the total processor resources. In our experiments we have used the SPECjvm98 benchmarks as a driving workload for our collector, effectively treating them as high-priority processes. The result is that when collection is on, it consumes about 50% of CPU resources.

While this seems high, there are two important points to note: first of all, the domain in which Java is likely to prosper is one in which the greater concerns are with development time and reliability, and less with CPU cost. A purely garbage-collection based real-time environment should have significant time-to-market advantages over the much more complex model of the RTSJ.

The second point is that the high-priority tasks are likely to have a much lower allocation rate, or could be programmed to do so. In that case, the percentage of the CPU that has to be allocated to the collector will significantly decrease.

Of course, this begs the question: are we better off with a simple programming model in which programmers have to adapt by reducing the allocation rate of some performance-critical code, or with a more complex programming model that gives them some tools (like scoped memory regions) for reducing this allocation rate. We believe that a simple, uniform, adaptable programming model is preferable.

5 Reducing Context Switch Times

The Metronome currently operates with a maximum pause time of 4 milliseconds, and using the current approaches we expect to drive this pause time to the sub-millisecond level. However, for some applications responses in the tens of microseconds are required. In this section we describe the features of the current collector that stand in the way of this goal, and describe how the design could be adapted to achieve these much lower pause times.

5.1 Priority Scheduling

First of all, the current system does not include any notion of high-priority real-time threads versus low-priority threads. Such a distinction would have to be incorporated, with the collector having a priority higher than the low-priority threads but lower than the high-priority threads. When scheduling the high-priority threads, a feasible schedule would have to include a time allotment for the collector thread to run, that was sufficient given the application threads' cumulative allocation rates.

This could easily be done by treating the addition of a thread with allocation rate a_i and maximum live memory m_i as the addition of two threads to the schedule: one is the thread itself, and the other is the additional work performed by the garbage collector.

High-priority real-time threads would be allowed to interrupt the collector thread, but the scheduling algorithm guarantee guarantee that the collector receives sufficient resources in each time period.

The modifications described below will allow much faster interruption of the collector.

5.2 Lazy Read Barrier

The first inhibitor to quick context switching out of the collector is an optimization of the read barrier which we call the “eager barrier”. In this form of the barrier, when a mutator thread loads a pointer onto the stack its forwarding pointer is immediately followed. This has the advantage that if the pointer is used in a loop, the read barrier is only executed once. However, it does mean that if the mutator is interleaved with the collector, and the collector moves objects, it must execute “fix-up” code on the stack frames to maintain the eager invariant (that is, that all stack references point to the current versions of objects).

By trading off some throughput for response time, we can employ the lazy version of the read barrier, which does not forward the pointer until it is used. In this case, there is no fix-up code required and any movement of objects by the collector does not inhibit context switch to the mutator. We measured the cost of this as about a 2% slowdown over the SPECjvm98 benchmarks.

Based on our experience we believe that the performance loss due to the lazy barrier can mostly be recovered, albeit at the expense of more complex compiler optimizations. Essentially, there is a spectrum between “eager” and “lazy”, and compiler optimizations can preserve the lazy property while reducing the number of forwarding operations.

5.3 Abortable Copy Operations

The next problem for context switching out of the collector is that it may be in the midst of a long-running atomic operation, in particular copying a large chunk of memory like a stacklet or an arraylet.

The solution is to make these operations abortable and restartable. The main difficulty is that the context switches must not be so frequent that the abort operations are able to impede forward progress of the collector. Thus the cost of the aborted operations must be factored into the collector cost.

5.4 Deferred Root Scanning

Probably the single largest inhibitor to rapid context switching out of the collector is the atomic operations performed when collection starts. In particular, the collector uses a “snapshot-at-the-beginning” technique. Thus, the roots in the thread stacks and global variables must be copied atomically.

Stacklets [6] allow the delay to be reduced by only requiring the snapshotting of the topmost stacklet of each thread, but this solution does not scale to large numbers of threads and introduces further problems because it requires that threads perform stack snapshot operations if they return from the topmost stacklet into a lower, un-snapshotted stacklet. Neither of these behaviors is acceptable in a high-frequency real-time environment.

The solution is to weaken the “snapshot-at-the-beginning” property. Instead we simply require that no reference from a stack be allowed to “escape” from the stack without being logged. Thus the write barrier, instead of recording just the old value (the Yuasa-style barrier) also records new values written from stack variables into the heap (the

Dijkstra-style barrier). In this manner, no references on the stacks can escape into the heap without being caught by the write barrier.

As a result, we both avoid the termination problems of the Dijkstra-style barrier as well as avoiding the need to complicate the read barrier (for instance by recording pointers loaded onto the stack during collection). Furthermore, even though the stacks may be changed by the mutators, we still only have to scan each stack exactly once (and can do so incrementally), since any relevant references not found in the stack scan must have been written by the write barrier.

This in turn allows us to interleave execution of mutators with root scanning, at a modest performance cost in the write barrier.

5.5 Safe Points

Finally, since our scheduler only performs context switches at safe points, there is the issue of delay introduced while waiting for threads to reach a safe point. In practice, safe points occur quite often. In order to meet real-time bounds, an analysis phase can be added which inserts extra safe points into large monolithic basic blocks.

6 Metronome versus RTSJ

In Section 2, we described some of the problems with RTSJ. Now that we have presented the Metronome, we examine the relative benefits of the two approaches.

The current collector can usually context switch in about 100 microseconds but in certain (short) phases of collection may take as much as 700 microseconds. With the modifications of Section 5, in particular the lazy read barrier and deferred root scanning, we expect to be able to bound the context switch time to 100 microseconds.

In choosing between RTSJ and the Metronome, one must balance greater control over memory usage and possibly superior performance with the greater programming effort. Metronome has the advantage of greater simplicity and retains the spirit of Java where the programmer is not burdened with memory management details: there are no scopes and no dynamic memory store or load exceptions.

RTSJ has the advantage that it may be possible to write a particular program to fit in a much smaller memory by careful use of `ScopedMemory`. However, using `ScopedMemory` might actually increase memory consumption because the size estimate is too conservative or if a large fraction of data within the scope dies during the scope's lifetime. RTSJ also has the advantage that context switching to `NoHeapRealTimeThreads` can be quicker than context switching out of the Metronome.

To capture the key benefits of both systems, we propose a hybrid system in which RTSJ is modified to take real-time GC into consideration. We propose removing the `NoHeapRealtimeThread` class so that the programmers do not have to program in a constrained fashion in which high-priority threads cannot access the heap at all. Instead, high-priority threads can communicate with lower-priority or even regular threads in the usual way. In the hybrid system, `MemoryAccessError` exceptions are eliminated because the read barriers associated with the `NoHeapRealtimeThread` are removed.

In this hybrid system, the expected development cycle begins with programming without scoped memory except where the fit is natural and intuitive. In prototyping the system, the programmer must then determine the memory characteristics of the program by some combination of analysis and profiling. These will result in establishing the number and size of scoped regions, the maximum live heap data, and the allocation rates of the various threads. From these parameters, the overall computational and space requirements of the garbage collector can be established and the feasibility of the entire system can be determined. Should the heap pose a problem, the programmer must reduce the allocation rate and live heap data by modifying the program logic or by increasing the use of scoped memory regions. What distinguishes this hybrid system from RTSJ is a less brittle programming model where memory requirements can be met by incrementally tightening the memory usage of the program. This is made possible by a greater reliance on the garbage collector. As mentioned before, the floating garbage possible with scopes can make scoped memory less attractive than garbage collection.

7 Conclusions

We have described the complexities of the RTSJ programming model, and shown that they will have an adverse effect on both ease of use and reliability, and may have adverse performance effects as well.

We have proposed an alternate approach to creating a real-time Java programming environment, which is based on constructing a true real-time garbage collector which is fully integrated with the scheduling system. This allows garbage collection to co-exist, and results in a much simpler programming model.

We believe that such a model is more consistent with the spirit of the Java language and will ultimately be more useful to the potential body of real-time Java programmers.

The techniques of the Metronome can also be applied to simplify the RTSJ specification by eliminating the need for threads that have no references to the heap, and eliminating an entire class of run-time memory access exceptions.

References

- [1] BACON, D. F., CHENG, P., AND RAJAN, V. T. Controlling fragmentation and space consumption in the Metronome, a real-time garbage collector for Java. In *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems* (San Diego, California, June 2003).
- [2] BACON, D. F., CHENG, P., AND RAJAN, V. T. A real-time garbage collector with low overhead and consistent utilization. In *Proceedings of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New Orleans, Louisiana, Jan. 2003), pp. 285–298.
- [3] BAKER, H. G. List processing in real-time on a serial computer. *Commun. ACM* 21, 4 (Apr. 1978), 280–294.
- [4] BOLLELLA, G., GOSLING, J., BROSGOL, B. M., DIBBLE, P., FURR, S., HARDIN, D., AND TURNBULL, M. *The Real-Time Specification for Java*. The Java Series. Addison-Wesley, 2000.
- [5] BROOKS, R. A. Trading data space for reduced time and code space in real-time garbage collection on stock hardware. In *Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming* (Austin, Texas, Aug. 1984), G. L. Steele, Ed., pp. 256–262.

- [6] CHENG, P., HARPER, R., AND LEE, P. Generational stack collection and profile-driven pre-tenuring. In *Proc. of the Conference on Programming Language Design and Implementation* (June 1998). *SIGPLAN Notices*, 33, 6, 162–173.
- [7] HIGUERA-TOLEDANO, M. T., AND ISSARNY, V. Analyzing the performance of memory management in RTSJ. In *The 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing* (Crystal City, Virginia, 2002).
- [8] YUASA, T. Real-time garbage collection on general-purpose machines. *Journal of Systems and Software* 11, 3 (Mar. 1990), 181–198.

Memory Management Design to the Concurrent Execution of RTSJ Applications*

M. Teresa Higuera-Toledano

Faculty of Computer Science, Complutense University of Madrid,
Ciudad Universitaria, 28040 Madrid Spain
mthiguer@dacya.ucm.es

Abstract. Our objective is to adapt the Java garbage collection to an embedded system, e. g., wireless PDA executing multimedia applications. We consider the concurrent execution of several applications within a single JVM, giving an approach to divide/share the memory among the applications executing concurrently in the system. Since a multimedia application is supported by several tasks, some of them being response time limited, while others being high performance oriented, we must consider the real-time characteristics of the GC. In order to have a different GC per application, we introduce and define application-specific memory, building upon the Real-Time Specification for Java (RTSJ) from the Real-Time Java Expert Group.

1 Introduction

Demands for multimedia services in embedded real-time systems, such as wireless Personal Digital Assistants (PDAs), are increasing. The use of PDAs is foreseen to outrun the one of PCs in the near future. However, for this actually happen there is still the need to devise adequate software and hardware platforms in order to not overly restrict the applications that are supported. In general, the environment must accommodate the embedded small-scale constraints associated with PDAs, and enable the execution of the applications traditionally supported on the desktop such as soft real-time multimedia applications that are becoming increasingly popular. In particular, it is mandatory to finely tune the management of memory consumption, and to enable provisioning new applications extending the capabilities of the mobile phones.

Addressing the above requirements then lies to devising resource management policies taking into account both real-time constraints and the available memory budget, and offering an open software environment which enables extending the application set that can be supported by the PDA. The ideal candidate to providing an open environment is Java, which appears as a major player in the area of embedded software environment, and allows us to get portable code, which can possibly be dynamically downloaded. Although, Java has some shortcomings regarding the target device, that have been solved by extending Java API to meet the requirements

* Founded by the Ministerio de Ciencia y Tecnología of Spain (CICYT); Grant Number TIC2002-00334.

appertained to embedded real-time software [4] and [11], such as real-time scheduling and predictable memory.

Therefore the need for executing multiple applications in the same JVM is increasing [2], [5]. When executing concurrently multiple applications, if an application consumes all the available memory, the other applications get starved. One way to avoid this problem is to divide the memory among running applications in the system, giving to each application a separate garbage collectable area. Hence, the partition of the heap in separate sub-heaps allows: invoking several collectors concurrently; having a collector per sub-heap that is customized according to the behaviour of the embedding application, minimizing the latency time to preempt a local collector from the CPU when a high priority task from another application arrives, and distributing the collector overhead among activities.

From a real-time perspective, the Garbage Collector (GC) introduces unpredictable pauses that are not tolerated by real-time tasks. Real-time collectors eliminate this problem but introduce a high overhead. An intermediate approach is to use Memory Regions (MRs) within which both allocation and de-allocation are customized and also the space locality is improved. Application of these two implicit strategies has been studied in the context of Java, which are combined in the *Real-time Specification for Java* (RTSJ) [4]. The J Consortium solution [11] proposes allocation contexts allowing us to group core objects that are free upon disposal of the allocation context. The core objects are not relocated and garbage collected, and core methods do not include code for synchronization with the GC.

This paper focuses on a memory management solution in order to divide/share the heap among different real-time applications accounting for the RTSJ.

1.1 The RTSJ Memory Model

The `MemoryArea` abstract class supports the region paradigm in the RTSJ specification through the three following kinds of regions: (i) immortal memory, supported by the `ImmutableMemory` and the `ImmutablePhysicalMemory` classes, that contains objects whose life ends only when the JVM terminates; (ii) (nested) scoped memory, supported by the `ScopedMemory` abstract class, that enables grouping objects having well-defined lifetimes; and (iii) the conventional heap, supported by the `HeapMemory` class. Objects allocated within immortal regions live until the end of the application and are never subject to garbage collection. Objects with limited lifetime can be allocated into a scoped region or the heap. Garbage collection within the application heap relies on the (real-time) collector of the JVM.

RTSJ also makes distinction between three main kinds of tasks: (i) *low-priority* that are tolerant with the GC, (ii) *high-priority* that cannot tolerate unbounded preemption latencies, and (iii) *critical* that cannot tolerate preemption latencies. Low-priority tasks, or threads, are instances of the `Thread` class, high-priority tasks are instances of the `RealtimeThread` class, and critical tasks are instances of the `NoHeapRealtimeThread` class. Synchronization among critical tasks and non-critical ones is problematic, since the non-critical may cause delays to the critical ones due to the execution of the GC. RTSJ solves the synchronization problem by introducing unsynchronized and non-blocking communication, e.g., the

`WaitFreeReadQueue` (resp. `WaitFreeWriteQueue`) class supports a *read-free* (resp. *write-free*) queue, which is unidirectional from non-real-time to real-time.

1.2 Paper Organization

The rest of this paper is organized as follows. Section 2 presents the general guidelines of our solution design to execute concurrently several applications. Section 3 details the characteristics of the local GC embedded in each application. Section 4 discusses resource management policies taking into account both real-time constraints and the available memory budget. Section 5 sketches an overview of related work. Finally, Section 6 offers some conclusions.

2 Extending RTSJ to Support Several Applications

In this section, the memory management model of RTSJ is extended to offer a multi-process execution. In the proposed solution, some memory regions are accessible by all the activities in the system, which allows inter-process communication by using both the communication model of Java based on shared variables and monitors, and the classes that the RTSJ specification provides to communicate real-time tasks and non-real-time threads.

2.1 Extending the MemoryArea Hierarchy

In order to obtain multi-process execution, we introduce the `MemorySpace` abstract class supporting three subclasses: the `CommonHeap` and `CommonImmortal` to support public memory without application access protection, and `ProtectedMemory` to define application-specific memory with access protection. There is only one object instance of both the `CommonHeap` and the `CommonImmortal` classes which are created at initialization system time and are resources shared among all activities in the system. In contrast, a new `ProtectedMemory` object is created when creating a new application and is a local resource protected from accesses of all the other activities in the system.

Creating a protected memory space implies the creation of both the local heap and the local immortal memory regions of the corresponding application. As in RTSJ, an application can allocate memory within its local heap, its immortal region, several immortal physical regions, several scoped regions, and also within the common heap and the common immortal spaces.

In order to obtain the reference of the common heap and the common immortal region in a similar way to the RTSJ model, we introduce the `instance()` method in both the `CommonHeap` and `ImmortalMemory` classes.

2.2 Sharing Memory

To facilitate code sharing, classes are stored within the common immortal space (i.e., the `CommonImmortal` object). In this way, all applications in the system access both code and data (i.e., *class variables*), of all classes. But there is a problem with the access to the class variables, declared as `static` in Java. These variables must be shared by all the tasks of an application, but they must be protected from the access of other activities. Thus, we maintain a copy of the class variables in the local immortal memory of the application. As in the solution given in [5], we maintain a copy of the class variables for each application using the class, while maintaining only a single version of the class code.

The same problem arises with *class monitors* (i.e., shared code related to synchronization), these methods are declared in Java as `static synchronized`. When a task enters a class monitor and is suspended by another task, if both tasks are from the same application, there is no problem. The problem arises if the two tasks are from different activities. To ensure mutual exclusion among tasks from the same application, while avoiding other activities to be affected, each application must maintain a separate copy of the monitor. The solution is then to allocate also in the immortal memory of the application a copy of the `static` code. This solution requires modifying the class loader to allocate in the immortal memory of each application, a copy of data and code declared with the `static` statement [5]. In general, there is a problem with resources shared among all tasks of an application that must be isolated from tasks of the other application.

2.3 Dealing with Critical Tasks

Whereas high-priority tasks require a real-time GC, critical tasks must not be affected by the GC, and as a consequence cannot access any object within the heap [4]. A reference of a critical task to an object allocated in the heap causes the `MemoryAccessError()` exception, which can be achieved by using *read barriers*. Note that read barriers occur upon all object accesses, which means upon executing both types of bytecodes: those causing a *load reference*¹ and those causing a *store reference*².

Note that read barriers are strictly necessary only when using read barrier-based collectors (i.e., incremental copying without handlers). When using a non-read barrier-based collector (i.e., mark-and-sweep, generational, copying-based using handles, and reference-counter), load operations do not affect the GC data structure. As consequence, the GC does not causes delays to critical tasks when accessing objects. Hence, we apply the same optimization as for the incremental GC which is to use write barriers instead of read barriers [2]. Since reads do not interfere with the GC, the restriction on critical tasks can be reduced to write barrier checks. And, the `MemoryAccessError()` exception, which raises when a critical task attempts to

¹ `getfield`, `getstatic`, `agetfield_quick`, `agetstatic_quick`, or `aaload` bytecodes.

² `putfield`, `putstatic`, `aputfield_quick`, `aputstatic_quick`, `aastore`, or `aastore_quick` bytecodes.

access an object within the heap, is changed by the `IllegalAssignmentError()` exception, which raises when a critical task attempts to assign an object that belongs to the heap [7].

3 Garbage Collection in an Embedded Multimedia Framework

Different GC techniques are appropriate depending on the real-time embedded application. If the application does not generate cyclic data structures, a reference-counting GC algorithm is the most appropriate. Alternatively, a generational GC collects the younger objects more frequently than the older ones. While real-time GCs provide the worst-case guarantees, generational GCs improve the average performance at the expense of the worst-case. Generational collectors may be good for some applications, which are soft real-time, like multimedia ones.

3.1 The Basic Collector Strategy

The characterization of the applications behaviour related with dynamic memory allocation helps choosing the local GC technique. There are also some important considerations when choosing a real-time GC strategy, among them are space cost and barrier costs. Copying GCs require doubling the memory space, because all the objects must be copied during GC execution. Non-copying GCs do not require this extra space, but are subject to fragmentation. We specifically consider the incremental non-copying GC based on the tri-color *treadmill* algorithm [1] that has been described in [7]. This algorithm allows the interleaved execution of the collector and the application³, which execution is synchronized by using write barriers to detect whether the application updates pointers. When the collection is completed, objects that must execute the `finalize()` method are moved to the finalize-list, which are executed by a specialized thread such as in [15]. Finally, for objects that have finalized, their memory is freed. Then, a compacting phase can be added to move objects into a continuous block into the heap, which implies some degradation of real-time guarantees.

Alternatively, we can use a generational copying collector such the employed by the Java *HotSpot* VM, which uses a GC based on the *train* algorithm [9] to collect the old space, which divides the old object space into a number of fixed blocks called *cars*, and arranges the cars into disjoint sets (*trains*). This technique uses write-barriers to trap both the inter-generation references (pointers from objects within the old generation to objects within the *new* generation), and references across cars within the same train.

We can improve the performance of the above collectors using the hardware write barrier support that the picoJava-II microprocessor [17] provides. From the standpoint of GC, this microprocessor checks for the occurrence of two type of write barriers: reference-based used to implement incremental collectors, and page-based designed specifically to assist generational collectors based on the *train-based* algorithm.

³ Each small GC unit interleaved with the application execution is called *increment*.

3.2 The GC and MR

Since objects allocated within regions may contain references to objects within the local heap, a tracing-based GC (e.g., incremental [1] or generational [9]) must take into account these external references, adding them to its reachability graph. When an object outside the heap references an object within the heap, we must add the object that make the reference to the root-set of the collector, which is achieved by using write barriers. When the collector explores an object outside the heap (i.e., a root), which has lost its references into the heap, it is eliminated from the root-set. Finally, when a scoped MR ends, all objects within the region having references to the objects within the heap are removed from the root-list of the collector, and all the objects within the region are moved to the finalize-list.

3.3 Running the Local GC

In RTSJ, an instance of the abstract `Scheduler` class implements a scheduling algorithm, providing flexibility to install an arbitrary scheduler. The `PriorityScheduler` subclass contains the base scheduling, which is preemptive and priority-based, having at least 28 real-time priority levels plus the 10 traditional Java threads priorities. Since fixed-priority scheduling with preemption is the most commonly used in real-time system construction, we adapt our real-time collector to this scheme by establishing the following main priority levels:

1. Low-priority task are mapped in the 10 Java priority levels.
2. The incremental GC is mapped in the lowest real-time priority level (i.e., the 11 priority level).
3. High-priority tasks are mapped in the lowest real-time priority levels, and can be interleaved with the GC (e.g., the 12-38 priority levels).
4. Critical tasks are mapped in the highest real-time priority levels, and can preempt the GC at any instant (e.g., the 12-38 priority levels).

When creating a task, in addition to the memory parameters, the memory region, and the runner code, we can specify both a `PriorityParameters` object and a `ReleaseParameters` object to control the task behavior. Figure 1 shows how we can create a task executing the incremental GC algorithm. Where the increment value is computed as the worst case execution time of a GC *increment* and is used to specify both parameters its *worst case execution time* and its *deadline*.

```
RealtimeThread gc = new RealtimeThread(
    new PriorityParameters(PriorityScheduler.getMinPriority()+10),
    new AperiodicParameters(increment, increment, null, null),
    null, null, null, new MyIncrementalGC());
```

Fig. 1. Scheduling the GC as a task in RTSJ.

4 Resource Management and Memory Negotiation

Resource management allows making real-time systems safe and extensible, and can be studied from three points of view: the ability to allocate resources to an application (*resource allocation*), the ability to track resource usage (*resource accounting*), and the ability to reclaim the resources of an application when it terminates (*resource reclamation*). There are two costs associated with the heap allocation, the direct cost of the `new()` method and the cost to perform the GC. In programs with high allocation rate, like multimedia applications, this cost can be substantial. Resource accounting is influenced by the way in which activities obtain services, and is difficult when there are shared system services. And resource reclamation presents some problems when a task terminates exceptionally.

4.1 Resource Allocation

The partition of memory allows us to invoke several collectors concurrently, where the reclamation rate can be different for each application. When an application allocates objects, the assigned memory is not necessarily continuous, e. i., partitions are not physical⁴. The execution of applications further relies on a negotiation protocol as commonly used when running multimedia applications. The handling of real-time constraints together with the limited capability of the PDA requires making sure that there is enough resource available to execute a new application. In the case where there is not enough resource left, it is common to have a negotiation protocol taking place between the application and the system where the application lowers its resource requirements by changing the resulting quality of service offered to the user. We are interested in addressing dynamic negotiation in order to benefit from resources left by applications that terminate.

Considering the RTSJ solution, prior to starting a task, a `MemoryParameters` object must be assigned to it. The memory requirements of a task are used for both the executive (to control admission) and the GC (to satisfy all tasks allocation rates). When a task exceeds its memory resource limitation, the executive can generate an exception to reject workloads. This solution allows communicating the memory requirements of a task to the system, but we can not change these requirements during the live time of the task. We found interesting that applications can negotiate dynamically with the real-time executive for resources. The resources budgets that we consider are: memory size and memory allocation rate. Upon arrival a new application, it must reserve resources. If there are not enough resources available, the application must revise its requirements. If still there are not resources available, the memory requirements of all applications can be revised. Finally, the incoming application will be either, accepted or rejected according to the results of the negotiation.

⁴ As an exception, the memory assigned to `LTMemory` objects must be continuous, because this kind of regions provides linear allocation time.

4.2 Resource Shared and Communication

Resource management is related with the communication model. The standard JVM supports a *direct sharing* communication model, which makes difficult resource reclamation (e.g., when an application ends and has shared objects with tasks of other applications). To communicate two tasks of different activities, both critical or both non-critical, we present a *limited sharing* model, where shared objects must be allocated within the common memory. Hence, this model is less flexible than direct sharing. However when an application ends, all objects are reclaimed without problems. For object allocated within the common spaces, class variables and class monitors are allocated within the common immortal memory. When applications have no address space shared, *copying* is the only possibility. This solution is the most flexible and the most adequate for a distributed real-time Java extension.

Communication among tasks which belong either to the same or different activities is safe, when both tasks are critical or when both tasks are non-critical. But communication when a task is critical and the other one non-critical require unsynchronized and non-blocking operations which make it unsafe. Note that when both tasks are from different activities, there is no problem with the local collector. However, the problem still exists with the common collector. Then, communication among critical and non-critical tasks of different applications must be achieved by using the common heap as the *memory area* parameter in the constructor of the *wait-free queues*.

4.3 Resource Reclamation

Each application heap is collected by a local collector (e.g., the GC described in Section 3), while the objects shared by the applications are collected by an incremental collector based on the reference-counting technique. Then, some form of distributed information to collect shared objects is required. We maintain a list of pointers, called *external-references*, to objects within the common heap having references from objects within other memory spaces. And for each object within the common heap, we maintain a counter giving the number of links to the object. An attempt to create a reference to an object into a field of another object requires three different treatments depending on the space to which the object belongs:

- Treatment A: the reference is within the common heap. It is not allowed for critical tasks, therefore the write must be aborted and the `IllegalAssignmentt()` must be triggered. For non-critical tasks, the write must take actions for the reference-counter collector, and also it causes the creation of an external-reference whether the X object is allocated outside the common heap (i.e., for inter-space references).
- Treatment B: the reference is within the common immortal memory. It is allowed and nothing needs to happen.
- Treatment C: the reference is within a protected. We distinguish two cases. For intra-spaces references (i.e., $\text{space}(X)=\text{space}(Y)$), we must take into account the assignment rules imposed by RTSJ [3] and actions needed by the local GC whether the reference is within the protected space [8]. For inter-spaces

references, the write must be aborted and the `IllegalAssignmentt()` must be triggered for inter-region references.

5 Related Work

The literature already provides us with a number of base solutions upon which we can build to meet our objectives. Focusing on work from the operating system community, relevant work on the issue of resource reservation has been examined in [14]. The proposed solution lies in a real-time JVM that interfaces with the abstractions of a resource kernel to build a resource-based JVM semantically compliant with RTSJ. Compared to this work, we are interested in examining management of resource consumption taking into account both real-time constraints and the available memory budget. Unlike traditional JVMs, in our proposed solution, the applications run concurrently within a single JVM instance, in a way similar to the Java Os from Utah [2]. Like in this solution, in order to provide secure and controlled accesses, we limit direct sharing among activities. When two activities want to communicate, they must share an object residing in the common heap. We take this solution as a trade-off between a more general solution such as allow activities to communicate using the RMI, and forbidding all possible communication.

The main applications for the PDAs are typically those run during a trip. In this context, applications will be Internet-based for accessing both discrete and continuous multimedia data. Internet services contain active code rather than static data, which raises serious trust and security issues. In order to provide strong isolation between services, both to enforce security and to control resource consumption. In [18], a new kernel architecture to isolate un-trusted applications has been proposed, whose function is to subdivide a physical machine into a set of fully isolated protection domains, and where each virtual machine is confined to a private namespace. Since Internet services are designed and operated by independent users, sharing is considered infrequent, and isolation has been strengthened.

A reconfigurable virtual machine supporting multiple user environments with varying degrees of criticality and privilege has been presented in [12]. This architecture provides hardware-enforced guarantees of resource separation, and it is based on the JEM-1 Java-based microprocessor. Hardware-based Java platforms (e, g., [6]) provide efficient support for bytecode execution, hard real-time, and also safe and secure multiple virtual machine execution.

In [10], we found a study of the requirements for embedded software environment aimed at wireless PDAs. These requirements have been addressed through a middleware platform, where the main issues in offering services for resource management lie in customized memory management. A solution to resource management and negotiation has not yet been addressed by the NIST group. However, its recommendation is to negotiate resources via the API. The PERC executive [13] analyzes availability of resources and determines when to accept a new real-time activity through a representation of the activity's *minimum* and *desired* resources, which allows resource negotiation.

6 Conclusions

This paper has presented a memory management design solution for extending the RTSJ specification to execute several activities concurrently in the same JVM. Two issues that we have addressed are how the memory is shared/divided among applications, and how allocation and deallocation are managed. To facilitate code sharing, classes are stored in the immortal common space. In order to provide secure and controlled access to the common memory regions and to maintain the assignment rules of RTSJ, we use a write barrier strategy. The logical distribution of the memory among applications can be based on a negotiation protocol, as typically used by multimedia systems. Regarding the customization of the collector running within each application heap, its selection depends on the features of the hosted application, and is left upon the responsibility of the developer (possibly aided by adequate analysis tools). For example, if the application does not generate any cyclic data structure, a reference counting GC algorithm is the most appropriate.

Our proposal builds upon existing work since the area of memory management. The contribution of our work comes from the adaptation and integration of relevant solutions in the context of RTSJ. This work need to be extended in multiple directions. Firstly, the proposed solution needs to be implemented and experimented with. Therefore, a memory allocation profile and techniques to determine the memory usage profile are very important issue requiring study.

References

- [1] H.G. Baker. "The Treadmill: Real-Time Garbage Collection without Motion Sickness". In Proc. of the Workshop on Garbage Collection in Object-Oriented Systems. OOPSLA'91. ACM 1991. Also appears as SIGPLAN Notices Vol. 27, no. 3, 1992.
- [2] G. Back, P. Tullmann, L. Stoller, W.C. Hsieh, and J. Lepreau. Java Operating Systems: Design an Implementation.. Technical report, Department of Computer Science, University of Utah, <http://www.cs.utah.edu/projects/flux>, August 1998.
- [3] W.S. Beebe and M. Rinard. "An Implementation of Scoped Memory for Real-Time Java". In Proc of 1st International Workshop of Embedded Software (EMSOFT), 2001.
- [4] G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. (*The Real-Time for Java Expert Group*). "Real-Time Specification for Java". RTJEG 2002. <http://www.rtj.org>
- [5] G. Czajkowski. Application Isolation in the Java Virtual Machine. In Proc. of Conference on Object and Oriented Programming, Systems Languages and Applications, pages 354–366. OOPSLA, ACM SIGPLAN, October 2000.
- [6] Hardin D.S. "Real-Time Objects on the Bare Metal". An Efficient Hardware Realization of the Java Virtual Machine. Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC). IEEE 2001.
- [7] M.T. Higuera, V. Issarny, M. Banatre, G. Cabillic, J.P. Lesot, and F. Parain. "Memory Management for Real-time Java: an Efficient Solution using Hardware Support". Real-Time Systems journal. Kluber Academic Publishers, to be published.
- [8] M.T. Higuera and M.A. de Miguel. "Dynamic Detection of Access Errors and Illegal References in RTSJ". In Proc. Of the 8th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS). IEEE 2002.

- [9] R. Hudson and J.E.B. Moss. Incremental Garbage Collection of Mature Objects. In Proceedings of the first International Workshop on Memory Management. September 1992.
- [10] V. Issarny, M. Banatre, F. Weis, G. Cabillic, P. Couderc, M.T. Higuera, and F. Parain. Providing an Embedded Software Environment for Wireless PDAs. In Proc. of the Ninth ACM SIGOPS European Workshop – Beyond the PC: New Challenges for the Operating System, September 2000.
- [11] J. Consortium Inc. “Core Real-Time Extensions for the Java Platform”. Technical Report. New-Monics Inc. http://www.j_consortium.org. 2000.
- [12] D.W. Jensen, D.A. Greve, and M.M. Wilding. Secure Reconfigurable Computing. Advanced Technology Center Advanced Technology Center. <http://www.klabs.org/richcontent/MAPLDCon99>
- [13] K. Nilsen. Adding Real-time Capabilities to Java. Communications of the ACM, 41(6), June 1998. pag. 49–56.
- [14] D. de Niz, R. Rajkumar. Chocolate: A Reservation-Based Real-Time Java Environment on Windows/NT. In Proc of Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000). June 2000.
- [15] Petit_Bianco and T. Tromey. Garbage Collection for Java in Embedded Systems. Proceedings of IEEE Workshop on Programming Languages for Real-Time Industrial Applications. December 1998.
- [16] Sun Microsystems. “The Java HotSpot Virtual Machine”. Technical White Paper. 2001. <http://java.sun.com>.
- [17] Sun Microsystems. “picoJava-II Programmer’s Reference Manual”. Technical Report. Java Community Process, May 2000. <http://java.sun.com>.
- [18] Whitaker, M. Shaw, and S.D. Gribble. Denali: A Scalable Isolation Kernel. In Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France, September 2002.
- [19] Wilson P.R. and Johnston M.S. “Real-Time Non-Copying Garbage Collection”. ACM OOPSLA Workshop on Garbage Collection and Memory Management. September 1993.

Experiences in Adopting Real-Time Java for Flight-Like Software

Edward G. Benowitz and Albert F. Niessner

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

{Edward.G.Benowitz,Albert.F.Niessner}@jpl.nasa.gov

Abstract. This work involves developing representative mission-critical spacecraft software using the Real-Time Specification for Java(RTSJ)[1]. Utilizing a real mission design, this work leverages the original flight code from NASA's Deep Space 1(DS1), which flew in 1998. However, instead of performing a line-by-line port, the code is re-architected in pure JavaTM, using best practices in Object-Oriented(OO) design. We have successfully demonstrated a portion of the spacecraft attitude control and fault protection, running on the standard Java platform, as well as on the reference implementation of the RTSJ. Our goal is to run on flight-like hardware, in closed-loop with the original spacecraft dynamics simulation.

In re-designing the software from the original C code, we have adopted OO techniques for flight software development. Specifically, we have taken advantage of design patterns[7], and have seen a strong mapping from certain patterns to the flight software. To ensure the correctness of measurement units, numerical computations are performed via an abstraction layer that checks measurement units at compile-time.

Our approach places an emphasis on pluggable technology. Interfaces, in conjunction with a façade pattern, expose only the behavior of an object, rather than exposing its implementation details. Since the RTSJ reference implementation does not currently support debugging, we chose to apply pluggable technology to the scheduler and memory allocation interfaces. Thus, real-time client code can be run on a standard Java virtual machine, allowing the code to be debugged in a graphical development environment on a desktop PC at the cost of decreased real-time performance. Once non-real-time issues have been debugged, the real-time aspects can be debugged in isolation on an RTSJ-compliant virtual machine.

1 Introduction

Java introduces a number of advantages for software development by improving software maintainability. We seek to take advantage of Java's features for flight software development. The RTSJ is a relatively new extension to Java which introduces real-time features. Since the specification is a new technology, there are few examples of real-time Java being applied to non-trivial problems. Our work involves creating real-time Java software which implements a realistic spacecraft control loop. In this section, we discuss the motivation for our work, the advantages of Java, and our approach.

1.1 Motivation

Flight software has a high development cost, due in part to the difficulty in maintaining the code. This lack of maintainability stems from the limitations of current implementation languages (typically C or C++). Memory can easily be corrupted due to the lack of pointer checking and array-bounds checking. Without operating system protections, these problems can occur as silent failures. Concurrency primitives are very low-level, and are not part of the language. In C, typical programs abound with error-prone switch statements and preprocessor directives, and only a globally shared namespace is available.

1.2 Advantages of Java

To address these issues, we are investigating Java as an implementation language for flight software. Java's maintainability improves developer productivity. According to [13,9], developer productivity is increased two-fold when comparing C++ to Java. Coding phase cost savings for a Java project versus a C++ project averaged nearly 40%. Java improves maintainability with its strong type-checking at both compile-time and run-time. Additionally, Java checks array boundaries, and ensures that variables are initialized. Standard Java provides automatic memory management, and Real-Time Java allows several forms of manual memory management where required (see 3.2 for details). Multi-threading and higher-level concurrency primitives are built into the language as well. Java can easily express pluggable components, as the notion of an interface is an explicit part of the language. Java provides for full encapsulation, and allows single inheritance with multiple interface inheritance. Java also provides extensibility through inheritance and dynamic class-loading.

Aside from the advantages of the language itself, the Java platform includes a large standard class library with support for most programming needs. Due to the large Java developer community, additional Java components are available from the internet, often for free. According to NIST[11], Java's higher level of abstraction leads to increased programmer productivity. The Java platform, coupled with Java language, improves application portability. Additionally, Java is easier to master than C++, and supports component integration and reuse.

1.3 Approach

We specifically chose to favor maintainability above all else during the architectural, design, and implementation phases of the development. Maintainability requires making extensive use of design patterns, taking full advantage of Java language features, using pluggable technology, and making appropriate use of commercial, off-the-shelf libraries and tools. During the performance evaluation phase, it is known that some maintainability will be sacrificed in order to optimize parts of the system to meet performance requirements. However, empirical evidence from systematic profiling of the application will dictate where and what is to be optimized, as opposed to prematurely attempting to optimize based on intuition, instincts, and/or assumptions of behavior.

For our implementation approach, we have chosen two subsystems to focus on. From the attitude control system, we have implemented a real-time control loop which detumbles a simulated spacecraft. In a detumble operation, a spacecraft with some initial set of angular velocities is controlled such that its angular velocity is reduced to a value close to zero. Our pure Java detumble control loop runs in real-time on both a standard Java virtual machine and on the RTSJ reference implementation. In addition, we have ported a C++ fault protection subsystem to Java, demonstrating a degree of autonomous behavior.

1.4 Tools

COTS graphical development tools were used extensively in this project. Specifically, the open-source Eclipse[6] integrated development environment provided graphical code editing, browsing, debugging, and refactoring capabilities. Headway's Review[10] product was used to graphically inspect our design, and allowed us to maintain a consistent architecture. Additionally, JProbe[16] was used to examine memory usage, and to identify critical regions for future optimization.

An RTSJ-compliant virtual machine is required for running real-time Java applications written to the RTSJ APIs. In addition to the the RTSJ reference implementation, several additional RTSJ implementations are now becoming available: JRate[3], OVM[12], JTime[17], JamaicaVM[8] and FLEX[14].

2 Pluggable Components, Factories, and Dynamic Class-Loading

Pluggable components are specified by an interface because implementation details are not visible from classes using the components. To hide the implementation class of a particular component, a factory is used instead of directly calling a constructor. The factory is tasked with constructing a particular instance of the pluggable component, and returning the component as an interface. An abstract factory provides a further step of abstraction. Each instance of an abstract factory can construct an instance of a pluggable component in a different fashion, typically instantiating a different implementation class.

We have used dynamic class-loading, in conjunction with abstract factories as a replacement for the C preprocessor. By using this approach, we allow the user of an application to choose at run-time the implementation corresponding to a particular interface. That is, the implementation of a pluggable component can be chosen at run-time. Clients of the interface use an abstract factory to request an instance of an interface. A particular implementation of an interface will have its own concrete factory as a subclass of the abstract class factory. The proper concrete class factory is dynamically loaded at run-time, returning the corresponding implementation of the interface. The advantage of this approach is that we can swap out implementations at run-time. Specifically, this was used to choose between our desktop scheduler implementation and the RTSJ scheduler implementation at run-time.

In the long term, dynamic class-loading has much greater potential for spacecraft missions. Current practice often requires reloading a binary image of the executable to a spacecraft, followed by a reboot. We envision that Java's dynamic class-loading facilities

could be used to provide additional functionality to a spacecraft by uplinking new class files to a running system, without requiring a reboot. This capability is outside of our current scope, but would be an interesting avenue for further research.

3 Real-Time Layer

There are several problems with traditional Java implementations that make it unsuitable for real-time programming, such as timing, scheduling, and memory management. The Real-Time Specification for Java addresses these issues by providing tighter semantics on scheduling, and by providing new memory management paradigms. We now discuss our wrapper layer for accessing these real-time services. We briefly introduce RTSJ's memory and scheduling features, and discuss some of the issues involved with the new memory management paradigm.

3.1 Scheduler

The nature of flight software requires that certain threads execute at certain times, and the times that the threads execute depends on the type of work being done. For instance, control loops run periodically while watchdogs run once at some time to signal the system of a potential problem. Defining the temporal boundaries and constraints for these threads is independent of the scheduling algorithm being used, but communicating these constraints to the scheduler is dependent on the scheduling algorithm and its implementation. We chose to apply the pluggable technology approach to our scheduler so that we can use whatever scheduling algorithm is available to us. Currently, we provide several varieties of scheduling requests:

- A one-shot timer. The scheduler will run a block of code after at a specific time.
- Periodic behavior: The scheduler will then run the block of code at the client specified rate.
- Standard: The scheduler will run the block of code when possible.

All of these behaviors can also specify a deadline that when crossed will cause a secondary block of code to be executed. Additionally we provide facilities for specifying a maximum percentage of CPU usage by a particular thread. The scheduler to be used is selected at run-time and instantiated through the use of dynamic class loading and factories.

Since the RTSJ reference implementation does not currently support debugging, our choice of pluggable technology allowed us to use the desktop for debugging. When running within an RTSJ-compliant virtual machine, our scheduler interface simply delegates out to the underlying RTSJ implementation. However, when running on a standard desktop Java virtual machine, the scheduler component uses our own implementation, written only using standard Java features. We emulate, as best as possible, the real-time scheduling features on a standard Java platform. Clients may chose between the RTSJ scheduler implementation and the desktop scheduler implementation at run-time.

Thus, real-time client code can be run on a standard Java virtual machine, allowing the code to be debugged in a graphical development environment on a desktop PC

at the cost of decreased real-time performance. Once non-real-time issues have been debugged on a standard Java VM, the real-time issues can be debugged in isolation on an RTSJ-compliant virtual machine.

3.2 Memory Areas

With the addition of the RTSJ's scheduling and memory management features, come new failure modes and programming pitfalls. The developer must consciously avoid violating memory area rules, and must ensure that no memory leaks occur.

Immortal Memory. Immortal memory is a new allocation scheme provided by the RTSJ. Once an object is allocated in Immortal memory, it is never freed. The advantage of this approach is that objects allocated in immortal memory have no need for interaction with the garbage collector. The disadvantage is that memory leaks are now possible. We recommend that allocations to immortal memory be performed in static initializers. We also require that object which are running in immortal memory only allocate in their constructors. With these restrictions in place, memory leaks can be avoided. However, this also places severe restrictions on which classes may be used. To use a JDK class while running in immortal memory, one must inspect the source code to ensure that allocations are only performed in the constructor. For the purposes of our own application, immortal memory was primarily used to load singleton factories.

Scoped Memory Usage. Scoped memory provides a means to dynamically allocate and free memory without using the garbage collector. Object allocated within a scope persist for the lifetime of the scope. Once the number of threads within a scope reaches zero, all objects allocated within the scope are destroyed. Additionally, scopes may be nested. The advantage for application programmers is that a large number of objects can be allocated and freed at once, without creating excess work for the garbage collector. One can think of scopes as a generalization of the C stack with the exception that the objects are finalized in the case of scopes.

A particular scoped memory region is represented by a scoped memory object, which itself must be allocated in a memory region. If one allocated a scoped region on the heap, the scoped memory object itself would be subject to interference from the garbage collector. For our application, all threads are created at application startup time. In this case, we can allocate scopes in immortal memory, and can create separate scopes on a per-thread basis. This scope allocation paradigm is quite similar to having one C stack per thread. The thread would then enter its own scoped memory, perform allocations, and then leave the memory area, automatically destroying the scope-allocated data. The size of the scope can be determined by profiling the memory usage characteristics of a particular thread, taking into consideration the requirements of the application and the available hardware resources.

The difficulty with scopes is their limited lifetimes. We enter and leaving a scoped memory region once for every iteration of our control loop. However, some data will need to persist beyond the lifetime of the scope, so we must provide mechanisms for copying data out of a scoped memory region. To facilitate this, we recommend providing

memory areas as parameters to factories. These factories can then be used to copy and construct objects in arbitrary memory areas.

4 Units

4.1 Problems with Past Practice

In current flight software projects, the measurement units are not explicitly part of the software. Perhaps measurement units are designated in an external document or in code comments, but there are no automated checks at either compile-time or at run-time to ensure that unit arithmetic is correct. For example, multiplying a velocity by a time should result in a distance. But since values are only represented as doubles, nothing prevents the developer from incorrectly treating the result of such an operation as a force, for example. We have already seen the disastrous consequences of incorrect units in the Mars Climate Orbiter mission.

4.2 Our Approach

To remedy this problem, we advocate making measurement units an integral part of the application code. Our package provides compile-time checking of measurement units. We provide interfaces for physical units, such as forces, distances, and times, and allow scalars, matrices, and tensors of values with physical units. With measurement units explicitly part of our code, we gain a number of advantages. Since measurement units are checked at compile-time, bugs are detected sooner, with a lower cost to repair them. Specifically, by using the units framework in our development, the detumble control loop was debugged in only 13 iterations. Because we knew that the measurement units were correct, pinpointing the actual cause of the errors became simpler.

In implementing our units framework, we have made use of COTS class libraries. Since units are pluggable components, alternative implementations are possible. For performing matrix and vector operations, we take advantage of the classes providing such functionality in Java3d. Additionally, for unit representation, we make use of the Jade library[4]. The admitted disadvantage of using Java for this situation is the lack of operator overloading, since the syntax for performing arithmetic does become quite verbose.

5 Conclusion

5.1 Summary

We have developed a pure Java prototype attitude control system, capable of performing a detumble maneuver in real-time, along with a pure Java fault protection subsystem. In developing this prototype, we have applied best practices in OO, including design patterns. We have demonstrated the applicability of pluggable components. Our measurement units framework allows the checking of unit correctness at compile-time. Based

on work with our units framework, we believe that operator overloading would be beneficial to real-time Java software development. We have explored the features of the RTSJ, discussing the usage of memory areas. We have created a pluggable real-time abstraction layer, enabled debugging on a standard Java platform.

Acknowledgments. This work was supported in part by the Center for Space Mission Information and Software Systems(CSMISS) at the Jet Propulsion Laboratory, and by the Ames Research Center.

References

1. G. Bollella , J. Gosling, B. Brosgol, P. Dibble, S. Furr, M. Turnbull, *The Real-Time Specification for Java*, Addison-Wesley, 2000.
2. J. Cooper, *The Design Patterns Java Companion*, Addison-Wesley, 1998.
3. A. Corsaro and D.C. Schmidt. "Evaluating Real-Time Java Features and Performance for Real-time Embedded Systems." Technical Report 2002-001, University of California, Irvine, 2002.
4. J.M. Dautelle, "JADE Java Addition to Default Environment", <http://jade.dautelle.com/>, 2002.
5. P. Dibble, *Real-Time Java Platform Programming*, Prentice Hall, 2002.
6. "Eclipse.org" , <http://www.eclipse.org/>, 2003.
7. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.
8. "Jamaica VM Realtime Java Technology", <http://www.aicas.com/jamaica.html>, 2003.
9. J. Jessup, "Java from the Horse's Mouth : An inside look at how Java has increased developer productivity", *Software Development*, pg. 73, March 1999.
10. "Headway Software", <http://www.headwaysoft.com/>, 2003.
11. NIST Special Publication 500-243, *Requirements for Real-time Extensions for the Java Platform: Report from the Requirements Group for Real-time Extensions for the Java Platform*, 1999.
12. OVM/Consortium, "OVM: An Open RTSJ Compliant JVM." <http://www.ovmj.org>, 2003.
13. E. Quinn and C. Christiansen, "Java Pays – Positively", IDC Bulletin #W16212, May 1998.
14. M. Rinard et al., "FLEX Compiler Infrastructure", <http://www.flex-compiler.lcs.mit.edu/>, 2003.
15. N. Rouquette, T. Neilson, and G. Chen, "The 13th Technology of DS1." *Proceedings of IEEE Aerospace Conference*, 1999.
16. "Sitraka JProbe", <http://www.sitraka.com/software/jprobe/>, 2003.
17. "TimeSys", <http://www.timesys.com>, 2003.

A Patterns Catalog for RTSJ Software Designs

Edward G. Benowitz and Albert F. Niessner

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{Edward.G.Benowitz,Albert.F.Niessner}@jpl.nasa.gov

Abstract. The Real-Time Specification for Java (RTSJ) [1] offers new features for scheduling and memory management, enabling new software design possibilities. In this survey, we provide descriptions of design patterns [5] for use with these RTSJ features. We discuss a pattern which integrates scoped memory usage with periodic real-time threads. The factory pattern is extended to accommodate RTSJ memory areas. Patterns for recycling immortal memory, and for extending the lifetime of a scope are presented.

1 Introduction

In this survey paper, we bring together current progress to date in identifying Design Patterns [5,6] for use with the Real-time Specification for Java(RTSJ) [1] in a format consistent with contemporary Patterns descriptions. We begin by discussing some elementary patterns for structuring real-time applications. Specifically, we discuss the use of scoped memory for applications with periodic real-time behavior. We then discuss a simple extension to the factory pattern to accommodate memory areas. We survey existing work on scoped memory allocation [4] and recyclable immortal memory [3], and discuss some of the implications of such approaches for software design.

Before describing the patterns in detail, we first define the following acronyms:

- RTT: Real-time thread
- NHRTT: No-heap Real-time thread
- RTI: Real-time interval. Each cycle of a periodic thread occurs in the time span of one RTI.

2 Scoped Memory Entry per RTI

2.1 Intent

Typical real-time application follow the following sequence of computations:

- Read sensors

- Run a control law, computing actuator commands
- Write actuator commands.

This sequence of computations occurs periodically, once per RTI. By integrating the use of scoped memory into this periodic behavior, excessive garbage generation can be avoided.

2.2 Motivation/Problem

If a control loop were implemented in pure non-RTSJ Java, allocations would be performed on the heap during each RTI. Much of the data allocated per RTI is immediately discarded, while only a smaller portion of the allocations made during the RTI contain data which will persist into the next RTI. Java's garbage collector, which adds non-determinism to an application's execution time and memory usage, would interfere with the application's real-time behavior. A real-time application developer needs a way to deallocate many objects at the end of an RTI, without requiring the assistance of the garbage collector.

2.3 Solution

The steps necessary to combine scoped memory, along with a typical periodic processing cycle, consists of the following steps:

- Enter a scoped memory area
- Read sensors
- Run a control law, computing actuator commands
- Write actuator commands.
- Copy data out to immortal memory or to the heap
- Leave the scoped memory area

2.4 Applicability

This pattern is applicable for real-time applications which make use of periodic threads. For the case of a real-time thread, data may be copied out to either the heap or to immortal memory. However, for NHRTTs, there is little choice but to copy data to immortal memory for storage between RTIs. Although scoped memory can be used equally well for non-periodic threads, the intent of this pattern is to show how memory deallocation via scopes can be integrated into a periodic processing cycle.

2.5 Static and Dynamic Structure

In Fig. 1, we show the dynamic structure of this pattern. From within a real-time thread which is scheduled periodically, the thread enters a scoped memory area, passing in a runnable interface. The scoped memory is entered, and calls the runnable's run method. Inside the run method, sensors are read, and the next

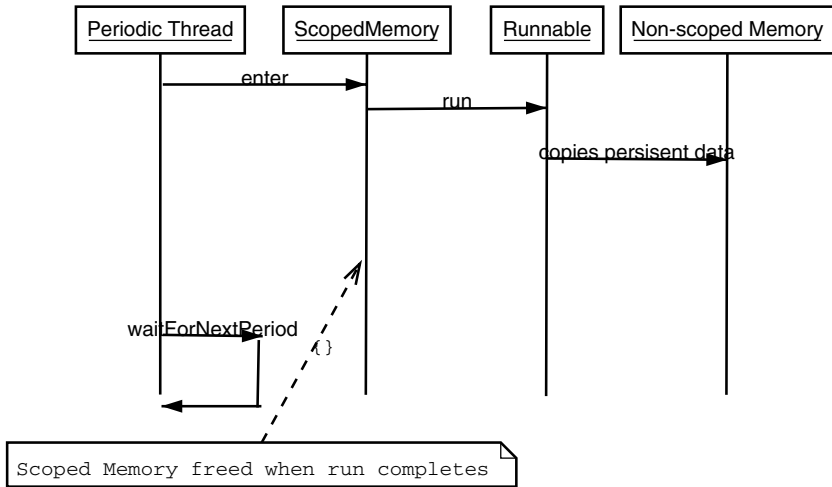


Fig. 1. Scope per RTI dynamic structure

set of actuator commands are computed. The runnable then stores any persistent data in a non-scoped memory region. When the run method completes, the scoped memory area is exited, and the memory allocated within the scoped memory region is then freed. Finally, the periodic thread will call `waitForNextPeriod`, blocking until the next RTI.

2.6 Consequences

The positive consequence of this pattern is that garbage generation is limited. However, by running in a scoped memory region, as soon as the RTI has finished, and the scope has been exited, all allocated data within that scope will be erased and unavailable. This means that without intervention, data cannot be stored between RTIs. Therefore, manual programmer intervention is required. The programmer must explicitly identify which pieces of data must have lifetimes longer than one RTI. Such data must then be explicitly copied from scoped memory to either immortal memory (for NHRTTs or RTTs) or to the heap (RTTs only).

2.7 Related Patterns

Because this pattern changes the default memory allocation context, this has consequences for other patterns used. Factory patterns may need to be augmented to use other memory areas besides the default allocation context. We discuss this extension to the standard factory pattern in section 3.

3 Using Factories with Memory Areas

3.1 Intent

Factories are used to allow the creation of objects implementing a specific interface, without having to disclose the implementation class. Aside from the intent for the standard factory pattern, the additional requirement is that factories be able to allocate objects implementing a particular interface in an arbitrary memory area.

3.2 Motivation/Problem

While using the traditional factory pattern [5] within the context of the RTSJ, all objects created by a factory will be created in the currently active memory area. Designers need the flexibility to specify in which memory area an object should be created. The use of `newInstance` is incompatible with the traditional factory pattern. Additionally, it is somewhat inconvenient to create a runnable and enter a memory area before calling a traditional factory's creation method.

3.3 Solution

For object creation methods within a factory, designs should add an additional parameter to the factory to specify the memory area in which the object should be allocated. Since the factory itself is a singleton, the factory should be put in immortal memory.

3.4 Applicability

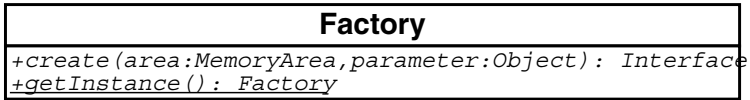
This is applicable to any RTSJ program wanting to program to interfaces rather than implementations, while maintaining flexibility in using memory areas.

3.5 Static and Dynamic Structure

Following the structure of the standard factory pattern, factories are singletons. The factory must provide methods which construct object, returning only the object's interface, thus preventing the exposure of the object's implementation class. The structure of the factory is shown in Fig. 2. Within the creation method, factories must either allocate the new object by using the memory area's `newInstance` method, or by entering the memory area first and then performing a new operation.

3.6 Consequences

Factory classes become dependent upon the RTSJ. The consequences of the standard factory pattern are also gained. Users are able to program to interfaces, removing the dependency on the particular implementation class. Factories remove the necessity of specifying which class to instantiate an object from.

**Fig. 2.** Factory

3.7 Related Patterns

This pattern is a slight modification to the factory pattern presented in [5]. Factories themselves are singletons. We observe that all instances of the singleton pattern should be allocated in immortal memory.

4 Wedge Thread

4.1 Intent

The developer may desire to prevent scoped memory reclamation until a later time. This is accomplished via the wedge thread pattern, introduced in [4].

4.2 Motivation/Problem

Typically, the memory associated with a scoped memory region used by one thread only persists for the duration of the enter method. After the scope has been exited, the memory can be freed. Developers, however may have the need to make use of data within the scoped memory area at a later point in their code, after the scope has been exited.

4.3 Solution

According to the RTSJ specification, scoped memory will not be reclaimed until all threads have exited the scope. To keep the scope beyond the entry and exit of the application's thread, an additional thread can be created, called a wedge thread, for the sole purpose of ensuring that the scope's memory is not reclaimed.

4.4 Applicability

According to [4], the wedge thread would be particularly applicable for storing an object reference in a scoped memory area's portal.

4.5 Static and Dynamic Structure

The wedge thread extends `RealtimeThread`, and is created with a higher priority than the application thread using the scope. When the scope memory needs to be kept alive, the wedge thread is started. The wedge thread proceeds to enter the scoped memory area, and waits on a lock. The methods in the lock are called

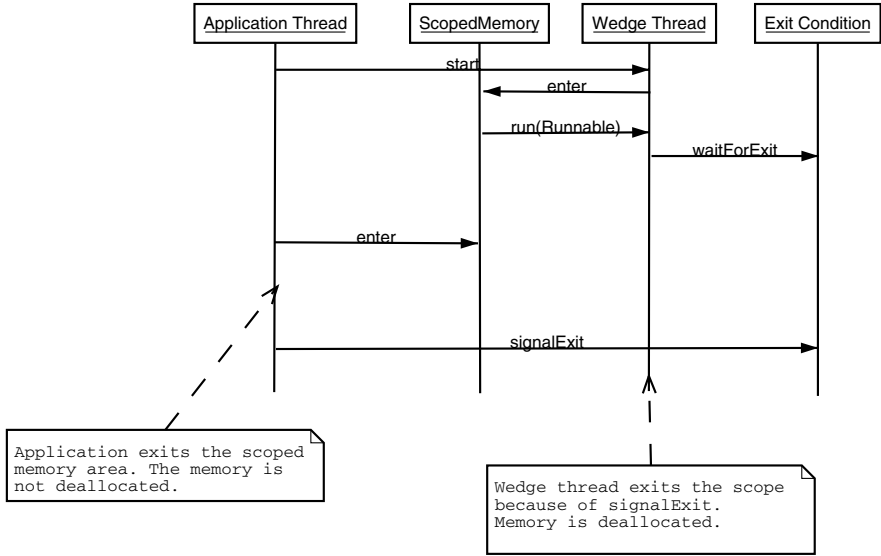


Fig. 3. Wedge Thread dynamic structure

waitForExit and signalExit. The lock is typically stored in the scoped memory area’s portal. The application thread can then enter and leave the scope, with the wedge thread ensuring that the memory remains alive. Finally, when the application is ready to deallocate the memory, the application thread notifies the lock object. This signals the wedge thread to exit the memory area, thus ensuring all threads have exited the memory area and allowing reclamation to take place. This is shown in Figure 3.

4.6 Consequences

The wedge thread allows extending the life of objects allocated within a scope beyond the life of one thread’s entry and exit of a scope. This comes at some cost. Firstly, there exists additional overhead at run-time of creating a higher-priority wedge thread. The wedge thread itself of course performs no computations; its sole purpose for existence is to provide an indirect way to manipulate the scoped memory area’s reference count.

4.7 Related Patterns

Additional patterns related to scoped memory usage can be found in [4].

5 Memory Pools

5.1 Intent

This pattern, described in [3], allows objects allocated in immortal memory to be recycled. This can be used to facilitate communication between NHRTTs and the outside world of Java threads.

5.2 Motivation/Problem

In the RTSJ, NHRTTs have a limited ability to touch memory. Specifically, NHRTTs will be very likely to allocate some objects in immortal memory. The lifetimes of the objects may not be known at creation time, such as the case where a NHRTT needs to export data to another thread. In such an instance, allocating objects in immortal memory will result in a memory leak. For such situations, a scoped memory area may not be appropriate, especially if the NHRTT needs to export objects having varying lifetimes. The need to recycle objects allocated in immortal memory becomes apparent.

5.3 Solution

A memory pool is a collection of pre-allocated object of a given class, typically created in immortal memory. When a developer needs to instantiate a new object of the desired class, the developer asks the pool to return an unused object of that class. In this respect the pool acts as a factory. However, when the developer is finished with the object, the object must be returned to the pool. This adds manual memory management and de-allocation to the RTSJ.

In one variation of this pattern, we might desire to express the freeing of a component as transparently as possible. In this case, we do not want to force users of a component to explicitly specify the pool from which the component was created. Therefore, we define an interface `Poolable`, which contains a `free` method. This allows an object to be de-allocated without the client having to know which pool to return the object to. This then forces the implementation object to store a reference to its pool within the pooled object's constructor.

5.4 Applicability

The recycling of immortal memory almost becomes a necessity when dealing with NHRTTs. This becomes especially important when allocating data to be passed into and out of wait-free queues, allowing the communication between NHRTTs and heap-using threads.

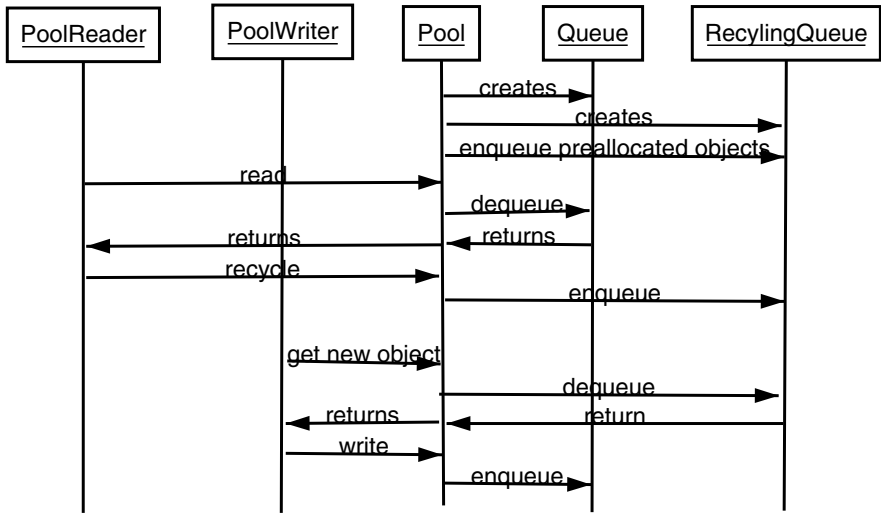


Fig. 4. Memory Pool

5.5 Static and Dynamic Structure

Dibble [3] recommends using two wait-free queues for communicating. This is one instance of using a memory pool. Memory pools can also be implemented independently of wait-free queues. However, wait-free queues are a likely candidate for being used in conjunction with memory pools, so they serve as an appropriate model.

We show the dynamic structure of using wait-free queues for memory pooling in Figure 4. First, the pool must be created. Internal to the pool, two wait free queues are created, the communication queue and the recycling queue. N elements are created in immortal memory, and are placed on the recycling queue.

When a reader is interested in the data, the reader first dequeues the data from the communication queue, and then examines it. When the reader is finished with the data, the data must then be recycled by placing the data in the recycling queue.

When a writer wants to enqueue data, the writer firsts dequeues memory from the recycling queue, and sets the data to the appropriate value. The writer then proceeds to enqueue the data on the communication queue.

5.6 Consequences

This pattern has the consequence of giving up a major advantage of Java: automatic memory management. Even the use of scoped memory offers some advantages over fully manual deallocation, in that memory deallocation takes place automatically. However, the introduction of memory pools takes Java a step backwards in terms of maintainability; developers must manually manage memory, deallocating objects at the proper time, reintroducing the possibility of

memory leaks. By using pools, memory leaks become more likely versus using immortal memory for truly immortal objects which require no explicit deallocation.

A second negative consequence of this pattern is that immutable objects may not be used in conjunction with memory pools. If an application's architecture already uses immutable objects, the application will have to be redesigned and re-implemented if it is to use memory pools.

A third disadvantage is that pools themselves must be created separately for each class. A single pool can only store fixed-sized objects of a single class.

5.7 Related Patterns

An alternative to this pattern is presented in section 6.

6 Memory Blocks

6.1 Intent

Memory pools must contain fixed sized objects of fixed classes for a given pool. The intent of the memory block pattern is to generalize the situation, by allowing a block of bytes to be allocated in immortal memory. Objects are written to and read from this memory block. The intent is to allow objects of varying sized to be pooled. In other respects, this pattern has the same intent as the Memory Pool.

6.2 Motivation/Problem

The motivation for this pattern is much the same as for using Memory Pools. This pattern presents a slightly different approach, however. Memory pools are limited to objects of a given fixed size and class. Each memory pool therefore contains objects of a single implementation type. Memory pools force architects to give up immutability, and forces exposure of the implementation classes. This patterns seeks to address the problem of importing and exporting data from immortal memory without sacrificing immutability.

6.3 Solution

Allocate a byte array in immortal memory. We call this a memory block. Objects being sent into the memory block are serialized into the memory block's byte array. Objects being received from the memory block are deserialized from the byte array. When an object is discarded, the memory block makes those bytes available for further allocations.

6.4 Applicability

This pattern is intended to be used for passing arbitrary objects in and out of a NHRTT through an immortal memory buffer.

6.5 Static and Dynamic Structure

A memory block is first created. Suppose a NHRTT is passing data out to a heap-using thread. The specific class that will be passed out of the NHRTT may vary over time. The NHRTT then passes the object to the memory block, which in turn calls the object's serialization method, putting the object in the memory pool's byte array. The memory pool is then sent out along a wait-free queue to a heap-using thread. The heap-using thread then asks the memory pool to deserialize the object, passing in the appropriate memory area. The memory pool then internally takes note of those bytes which are no longer in use.

6.6 Consequences

An application designer can re-use immortal memory while allowing application designers to pool objects of different classes and sizes. Immutable objects can then be used in designs, provided that they can be serialized in some form to a byte array. The disadvantages are still significant. We have reduced Java programming in essence to storing and retrieving data from a mechanism not unlike a Fortran common block. Additionally, the application must pay for the serialization and deserialization costs. Using `java.io` for serialization in the context of a NHRTT is suspect unless the developer has analyzed the serialization implementation to determine its memory allocation characteristics. Some custom serialization may be necessary. Fragmentation can occur in the memory block since objects can be of different sizes. In the end, the user is forced to manually write a memory manager.

6.7 Related Patterns

An alternative to this pattern is presented in section 5.

7 Conclusion

We have provided a survey of some of the known patterns for using the RTSJ. The RTSJ introduces new memory allocation techniques, and we have seen new approaches to dealing with the facilities. As developers adopt to new features, they should be aware of the consequences of using RTSJ's features for their architectures. Certain patterns limit an application's architecture, eliminating the possibility of using immutable classes. Although the RTSJ brings powerful features for working with real-time constraints, there is no free lunch: the RTSJ, and by extension some RTSJ patterns, introduce new dimensions for developers to consider, representing a necessary departure from Java's hallmark simplicity.

Acknowledgments. This work was supported in part by the Jet Propulsion Laboratory.

References

1. G. Bollella , J. Gosling, B. Brosgol, P. Dibble, S. Furr, M. Turnbull, *The Real-Time Specification for Java*, Addison-Wesley, 2000.
2. J. Cooper, *The Design Patterns Java Companion*, Addison-Wesley, 1998.
3. P. Dibble, *Real-Time Java Platform Programming*, Prentice Hall, 2002.
4. J. Fox, D. Holmes, F. Pizlo, J. Vitek, “Scoped Memory: Design Patterns and Semantics”, *Proc. 24th IEEE International Real-Time Systems Symposium*, 2003.
5. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.
6. D. Schmidt, M. Stal, H. Rohnert, F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects Volume 2*, Wiley and Sons, 2000.

RTSJ Memory Areas and Their Affects on the Performance of a Flight-Like Attitude Control System

Albert F. Niessner and Edward G. Benowitz

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

{Al.Niessner,Edward.G.Benowitz}@jpl.nasa.gov

Abstract. The two most important factors in improving performance in any software system, but especially a real-time, embedded system, are knowing which components are the low performers and knowing what can be done to improve their performance. The word performance with respect to a real-time, embedded system does not necessarily mean fast execution, which is the common definition when discussing non real-time systems. It also includes meeting all of the specified execution deadlines and executing at the correct time without sacrificing non real-time performance. Using a Java prototype of an existing control system used on Deep Space 1[1], the effects from adding memory areas are measured and evaluated with respect to improving performance.

1 Introduction

1.1 Goal

Many features are added to the Java virtual machine and libraries through the Real-time Specification for Java[2] (RTSJ) with the intent to improve the real-time performance of Java. One of the more interesting features is the addition of scoped memory, where the intent is to separate functional regions of the user's application from interactions with the garbage collector (GC). In theory, a scoped memory area is entered prior to the execution of a functional region and exited after the region. The functional region does not interact with the GC because reclamation occurs when the thread count in a scoped memory region goes to zero and is performed without the aid of the GC. Of course, allocating and reclaiming memory without the aid of the GC means that there are rules on cross referencing memory areas that are detailed in the RTSJ. The goal of this work is to demonstrate that the intent of the scoped memory area holds true with "real world" software.

1.2 Approach

A Java prototype of the Deep Space 1 attitude control system was developed in order to show that Java is usable in the production of spacecraft software. The

prototype was profiled on the desktop to measure memory and processor usage. The profile information shows that almost all of the garbage collected memory is allocated in the portion of the system that computes the output response from the sensor input – the control law. The excessive allocation is due to the choice of architecture for the prototype which forces an immutable implementation of a physical units package. The physical units package is used extensively when converting from the input of angular velocity to the command thrust of each individual thruster. Hence, the control law allocates enough data to activate the GC in about 3 cycles. The software was then adapted to include placing the control law within a scoped memory block whose size was larger than required as measured from the profiling and placing instrumentation in critical parts of the system. The latter adaptation was done because there are no tools available for profiling with Timesys's Reference Implementation[3] of the RTSJ (RI).

1.3 Tools

COTS graphical development tools were used extensively in this project. Specifically, the open-source Eclipse[4] integrated development environment provided graphical code editing, browsing, debugging, and refactoring capabilities. Headway's Review[5] product was used to graphically inspect our design, plan refactoring, and allow us to maintain a consistent architecture. Additionally, JProbe[6] was used to examine memory usage and to identify critical regions for future optimization. Lastly, Timesys's Reference Implementation[3] of the RTSJ is used because it is the only fully functional, freely available implementation of the RTSJ.

2 Measuring Time

The original intent was to use the clocks and date functions provided by Sun's JVM and the RI. However, both of these implementations limit the time resolution to a millisecond and the smallest time scales being measured are hundreds of microseconds. Therefore, a simple JNI interface to the Linux time services was developed which has a microsecond resolution. This section is devoted to the characterization of the JNI interface with Sun's JVM and the RI, using the same shared libraries and byte code for all tests.

2.1 Using Sun's JVM 1.4.1

As can be seen from the histogram (see Fig. 1), using either the Mandrake 9.0 stock kernel or the Timesys real-time Linux kernel version 3.2 has little affect on the latency where almost all calls are between 3 and 25 microseconds. It can also be seen in Fig. 3 that there appears to be a periodic delay associated with the Sun JVM which corresponds to the third mode around 1.3 milliseconds in the histogram. An attempt was made to profile this test in order to determine if the third mode was related to the GC, but the profiler measurement interfered

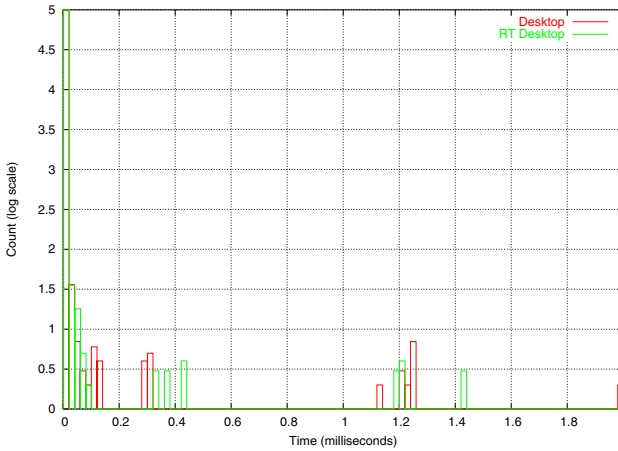


Fig. 1. JVM 1.4.1 Latency Histogram. This diagram illustrates the latency behavior differences between the Mandrake 9.0 stock kernel and Timesys’s real-time kernel with respect to retrieving the current time using the JNI and a call to `gettimeofday` from `jsys/time.h`.

with the test and the results were less than clear. However, an analysis of the test software indicates with a good level of confidence that the object allocation rate corresponds to the second mode, but the analysis is not complete and is too speculative to say conclusively that it is the GC. There are only minor differences in results between the two kernels and they all appear in the distribution of higher ordered modes.

2.2 Using the RI

As can be seen from the histogram (see Fig. 2), the RI behaves wildly different from the Sun JVM, where the main difference for the little bit of code being executed is the HotSpot compiler. Given a very aggressive GC, the allocation rate links the second or third mode to the garbage collector. However, there is still an undetermined, systematic error just as influential as the GC which is causing the other mode. The large difference can be seen in the first mode of the histogram, where the RI’s distribution is much more Poisson than is Sun’s JVM counterpart¹ and is centered at a much higher value. The best explanation for the constant offset and wider distribution is the RI’s lack of HotSpot or any other just-in-time compiler technology. Each iteration of the loop in the RI is interpreted, while only the first few in Sun’s JVM are interpreted before they are compiled. However, expecting transient delays from the HotSpot compiler, the first 100 were ignored in all the tests, allowing the JVMs to reach a steady-state condition before measuring began. Hence, none of the delays seen should be associated with HotSpot.

¹ It is expected that the first mode of the histogram for Sun’s JVM is also Poisson, but at a much smaller scale like 2 to 50 microseconds.

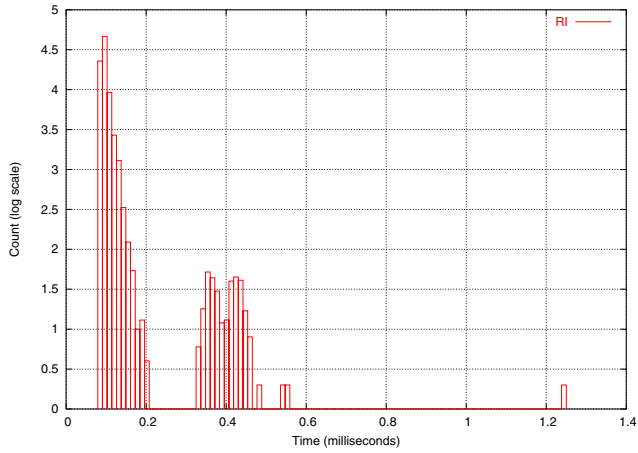


Fig. 2. RI Latency Histogram. This diagram illustrates the latency behavior of the RI with Timesys’s real-time kernel with respect to retrieving the current time using the JNI and a call to gettimeofday from `jsys/time.hi`.

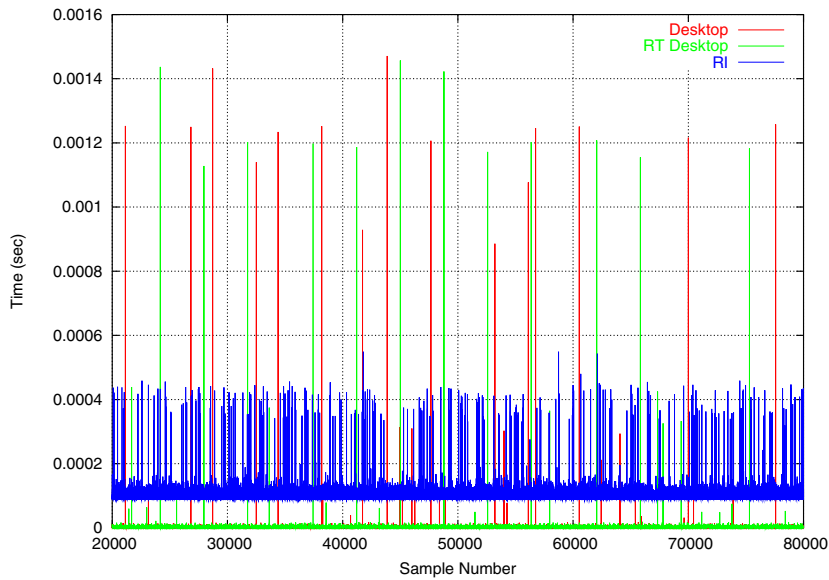


Fig. 3. Latency Times. This diagram is the measured latency from a call through the JNI to gettimeofday from `jsys/time.hi` for all JVM’s and kernels used. It is the data that was used to construct the histograms in figures 1 and 2.

3 Measuring Performance

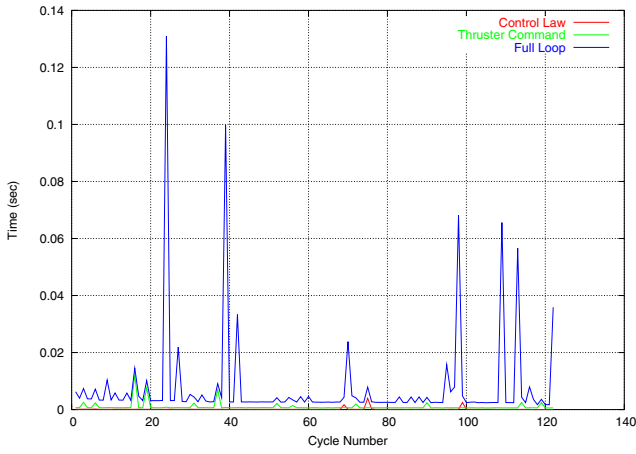
The Java prototype being measured can be broken into three distinct parts: the full software loop, the control law, and the thruster command processing. The thruster command processing portion takes as input the desired thrust from the control law and commands the individual thrusters on the spacecraft to exert the specified force. The control law takes as input the latest sensor readings and the desired attitude requirements, allowing it to compute the desired force to meet its requirements. The full loop does all of the management related tasks, making sure that the other two components have their necessary inputs and that all other management tasks are made aware of what is happening.

Just as with the previous tests, the software is moved from a Mandrake 9.0 system with a stock kernel to the Timesys real-time kernel using Sun's JVM as a baseline. It is then tested with RI JVM using none of the RTSJ features. The RI scheduler is then added, followed by scoped memory areas. Some of the results will then be compared to highlight the most interesting details.

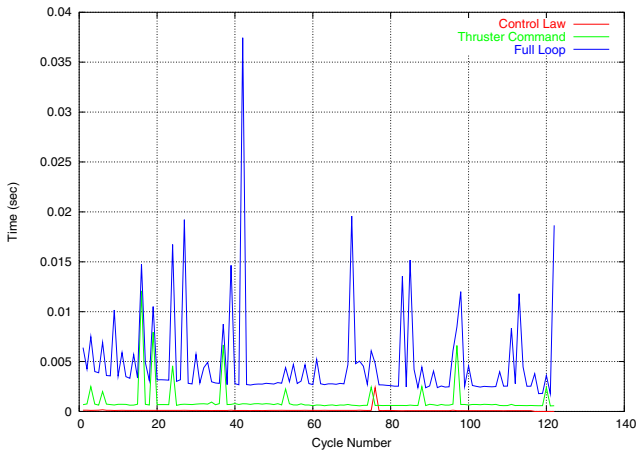
In order to execute the same code on both the desktop and with the RI, an abstraction layer was added to the system to separate the problem being solved from any implementation of the desired solution. There are two abstraction layers that are used during this test to change the behavior of the Java prototype: a scheduler abstraction layer and a memory area abstraction layer. The scheduler abstraction layer basically delegates either to the RI default scheduler or to a home-grown scheduler. The home-grown scheduler was written not as a serious scheduler but, rather, as a tool to allow the Java tools available on the desktop to be used with the prototype. The memory area abstraction layer either delegates to the RI memory areas when on that platform or simply uses the heap when using Sun's JVM, and the scoped memory is allocated in the immortal space when delegating to the RI.

3.1 On the Desktop

The performance chart (see Fig. 4) shows a distinct improvement (roughly 4x) in the maximum processing time between the stock kernel and Timesys's real-time kernel. In any case, all cycles were processed within their allotted time. It is also important to notice that none of the delays over two milliseconds occurred while the control law was in operation, which is the heaviest allocation portion of the system. Profiling also suggests the GC caused the difference, but it cannot be stated conclusively. If it really was the GC, the expectation is the GC would activate during the heaviest allocation portion of the system because memory was exhausted. Otherwise, the processor duty cycle is less than 25%, which leaves plenty of processing power for the GC to complete. If the GC did not complete in time, then the software would either process as normal or report that a complete cycle was missed. The jitter from the GC and the scheduler is ignored as long as the cycle boundaries are not exceeded.



(a) stock kernel



(b) Timesys real-time kernel

Fig. 4. Desktop Performance. This diagram illustrates, when comparing (a) to (b), the change in performance due to kernel changes. The control law takes such a small portion of the time that it is always near zero. The processing of the results and other management overhead is where the variations in processing time occur. Otherwise, the real-time kernel is somewhat slower, but otherwise performs like the stock Mandrake kernel.

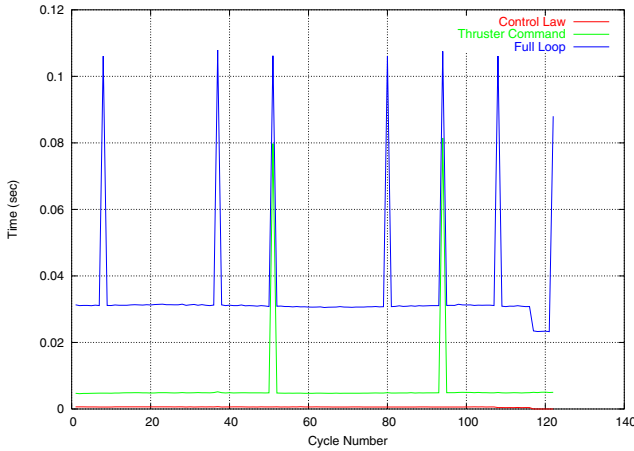


Fig. 5. RI Performance. This diagram illustrates, when comparing it to Fig. 4(b), the difference between the RI and Sun’s JVM 1.4.1. The overall performance is slower, but seemingly more deterministic.

3.2 Using the RI

The performance chart (see Fig. 5) shows that the RI, even without memory areas and the built-in scheduler, is much more deterministic than Sun’s JVM. The 100 millisecond spikes in the loop are interactions with the GC, and those interactions that are missing occurred in portions of the system that are beyond the scope of this paper – namely, the home-grown scheduler developed for the desktop and the spacecraft simulator. It is interesting that none of the GC delays occurred within the control law itself; the reason has still not been identified.

3.3 Using the RI Scheduler

The home-grown scheduler was then replaced with the RI scheduler. The performance chart (see Fig. 6) shows the full interaction of the system with the GC – the 120 millisecond spikes. Three of the GC executions occurred while commanding the thruster. There is a small bit of allocation (about 5% of one cycle) that takes place between the completion of the control loop and the start of the thruster command. Hence, it is not possible to determine if the GC execution occurred before or after the thruster command in the other cycles. However, the results from adding memory areas would imply that the delays come after the thruster command is executed and some cycle cleanup is in progress.

3.4 Adding Memory Areas

Lastly, scoped memory areas were included in the test system. The performance chart (see Fig. 7) does not contain all of the GC interactions, which is a minor

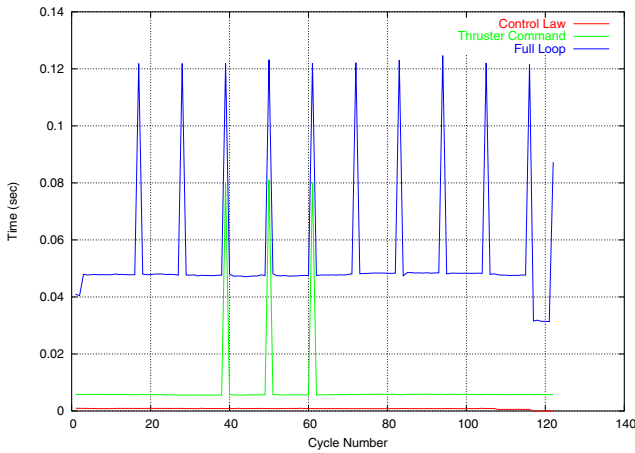


Fig. 6. RI Scheduler Performance. This diagram illustrates, when comparing it to Fig. 5, the performance changes from using the RI default scheduler.

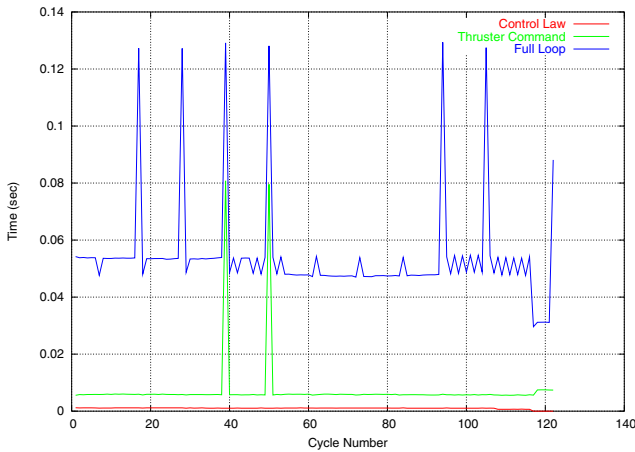


Fig. 7. RI Memory Area Performance. This diagram illustrates, when comparing it to Fig. 6, the performance changes from using the RI default scheduler and memory areas.

mystery. The RI scheduler is being used, but there is a more processing taking place because of the scoped memory area. Hence, it is not too surprising that the interaction between the test system and the GC changes. There are minor spikes at the correct periodicity to be the GC is another minor mystery. There is no reason to measure such minor delay from the GC unless the GC only did a partial reclamation before returning control to the test software since the threads are all at the normal priority and none of them are no-heap real-time threads.

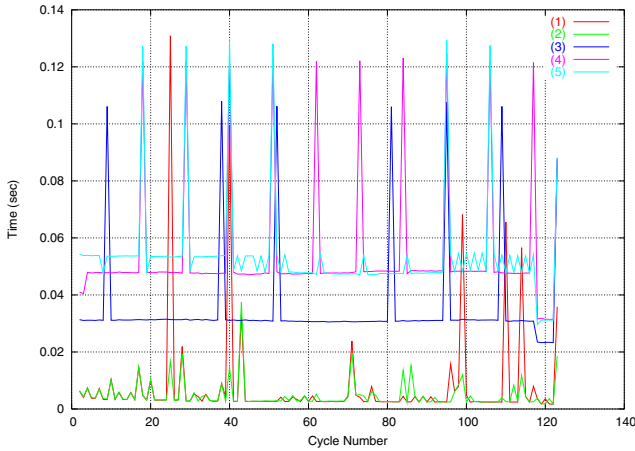


Fig. 8. Full Loop Comparison. (1) is Sun's JVM 1.4.1 with Mandrake's 9.0 stock kernel. (2) is Sun's JVM 1.4.1 with Timesys's real-time kernel. (3) is the RI with the same home-grown scheduler used in (1) and (2) and no RTSJ other features either. (4) is the RI with its default scheduler and no other RTSJ features. (5) is the RI with its default scheduler and memory areas covering the control law. This diagram simply collects the data from the previous diagrams for a more direct comparison. Not all of the lines are easily visible as (1) and (2) coincide nearly everywhere as do (4) and (5).

3.5 Comparing the Results

Comparing the full loop performance of each platform (see Fig. 8) highlights four very interesting details. First, the best loop time, even if it is less deterministic, is using Sun's JVM with Timesys's real-time kernel. Second, the loop processing time increases as RTSJ features are used. Third, the periodicity of the GC changes with schedulers – (3) and (4) in Fig. 8 – and the period is larger with the home-grown scheduler even though the allocation rate and volume are larger. Fourth, the periodicity of the GC activity is the same between (4) and (5) in figure 8, even though there is at least a 30% reduction in allocation on the heap.

The fact that Sun's JVM performs better than the RI in these tests is not too disturbing. To put this observation into perspective we need to investigate three anomalies: One, the large variation between (1) and (2) in Fig. 8 can be attributed to the difference in background services that were running simultaneously, which could have interfered with the test; most notably, X11 was not active with the Timesys real-time kernel. Two, the main difference between (2) and (3) in Fig. 8 is just-in-time compilation technology that is present in Sun's JVM and non-existent in the RI. Third, the test does not stress the capabilities of the hardware that the test was performed on; which is to say, the load is less than 50% of the hardware's capability. Hence, while Sun's JVM did better in this instance, it probably would not scale with loading which a full implementation of the RTSJ presumably would. This observation does imply that the

performance, both in the real-time and non real-time sense, can be significantly improved from compilation.

Using RTSJ features adds performance penalties. Increased time with the addition of memory areas is understandable because more checks are required in order to detect illegal assignments across memory barriers. The increase between choices of schedulers is a bit more perplexing. The home-grown scheduler is a very poor scheduler that uses very short sleep intervals, the smallest available period, as interrupts for yielding control to the rest of the test system. The sleep interval is notoriously poor with jitter and this, of course, bleeds over to the home-grown scheduler, which is why I called it very poor. However, the difference between (3) and (4) in Fig. 8 indicates the home-grown scheduler requires less processing power than the RI's scheduler, which surprised me because the home-grown scheduler is poorly written, is a high allocation rate, and is interpreted. The RI scheduler, on the other hand, appears to be part of the binary distribution and is interfaced through the JNI, which suggests that the scheduler should be fast and efficient. Hence, the penalty observation implies that one should measure every feature before using it to improve performance because it may not have the expected outcome.

The home-grown scheduler GC period is smaller than the RI scheduler's period. Again, since the RI's default scheduler appears to be a binary distribution accessed via the JNI, then less allocation should be taking place. Less allocation means that it would take more cycles to allocate enough trash to activate the GC. The incorrect change of periodicity is simply an extension of the penalty observation and implies the same consequences as well.

Significantly reducing heap allocations through the use of scoped memory increases overall processing time and the GC's activation periodicity does not change. This observation is contrary to the intent of scoped memory regions and is more than just an extension of the penalty observation. As Fig. 9 shows, the processing time of the control law is affected by the GC when running with Sun's JVM, but the RI performance contains all of the same features which are more than one cycle in width with only a constant between them. The control law is straight forward code with a single branch that is clearly present around cycle 110, where the type of compensator is changed. In the two cases (3) and (4) in Fig. 9, it is believable that they both have the same features since they both use the heap. However, (5) in Fig. 9 has the same structure as (3) and (4), which implies one of the following:

1. The structure is a function of reading the time through the JNI and therefore appears in all the RI runs. Fig. 3 contradicts this implication as there is no frequency of delay and, since the structure is defined by more than a single cycle, it is unlikely for a random process to repeat so well over an extended time.
2. The use of the scoped memory area is erroneous and the heap is being used. This is unlikely because an illegal assignment exception had to be fixed prior to the test working.

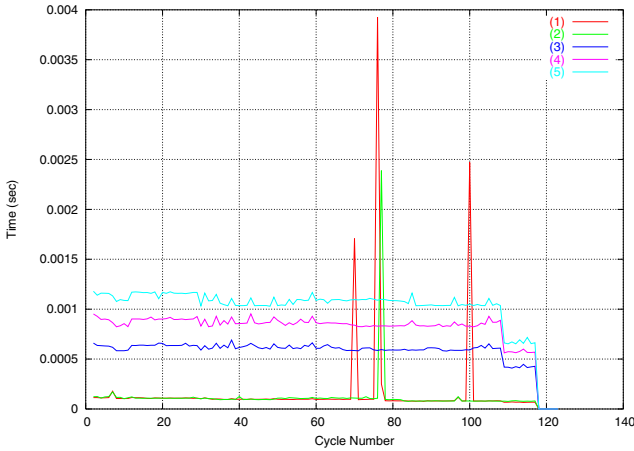


Fig. 9. Control Law Comparison. See Fig. 8 for line definitions. This diagram simply collects data from the previous diagrams for a more direct comparison. In this diagram, only (1) and (2) are nearly coincident.

3. Scoped memory is strongly related to the heap and therefore exhibits some of the same features, but, at the end of the day, it is independent of the heap and the intended benefit can be realized. Fig. 8 clearly shows that the scoped memory area and the heap are not independent because the periodicity of the two tests, (4) and (5), are identical when there is at least a 30% reduction of allocations to the heap.

Since none of the implications are valid, the intent of the scoped memory area cannot be realized. The last observation implies that memory areas are not very effective in reducing GC interaction with the user's application.

4 Conclusion

The best performance enhancements came from compilation and not from trying to isolate the system from the GC through the use of memory areas. Memory areas require additional run-time processing time to ensure that dangling references and other problems do not occur. The use of static compilers[7,8] with automated scoped memory detection[9] would remove the necessity for some of the run-time checking, and, perhaps, allow the intent of scoped regions to be realized.

Also, the lack of a tool API specification in the RTSJ and/or performance monitoring tools themselves makes it near impossible to gain conclusive data from the RI. If the RTSJ provided a required tools API for compliant JVMs, then the developer could use a generic tool or, in the worst of conditions, develop the required tools when manufacture does not supply them. The API would

have to give the user visibility into GC, memory areas, the scheduler, and event handling.

As an aside, one of the problems associated with using the RTSJ memory area is the difficulty of moving data from one memory area to another; it has a viral affect on the design and architecture of the software (for further details see [10]). It was particularly time consuming and tedious to add memory areas in an architecture that uses the immutable object as a way of improving thread safety.

As a further aside, this is a single test case and its performance with RTSJ features may be improved by removing many of the abstraction layers that separate the prototype from any specific real-time extension to Java and fully embracing the RTSJ and its features, the existing benefits of the Java prototype would be compromised through increased complexity and the loss of the architectural, design, and maintenance benefits of Java.

Acknowledgments. The research in this paper was supported by and carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA.

References

1. N. Rouquette, T. Neilson, and G. Chen, "The 13th Technology of DS1." *Proceedings of IEEE Aerospace Conference*, 1999.
2. G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, M. Turnbull, *The Real-Time Specification for Java*, Addison-Wesley, 2000.
3. Timesys Reference Implementation, http://www.timesys.com/index.cfm?bdy=java_bdy_ri.cfm, 2003.
4. "Eclipse.org", <http://www.eclipse.org/>, 2003.
5. "Headway Software", <http://www.headwaysoft.com/>, 2003.
6. "Sitraka JProbe", <http://www.sitraka.com/software/jprobe/>, 2003.
7. M. Rinard et al., "FLEX Compiler Infrastructure", <http://www.flex-compiler.lcs.mit.edu/>, 2003.
8. A. Corsaro and D.C. Schmidt. "Evaluating Real-Time Java Features and Performance for Real-time Embedded Systems." Technical Report 2002-001, University of California, Irvine, 2002.
9. Morgan Deters and Ron K. Cytron, "Automated Discovery of Scoped Memory Regions for Real-Time Java." In *Proceedings of the 2002 International Symposium on Memory Management* (Berlin, Germany), pp. 25–35. ACM, June 2002.
10. P. Dibble, *Real-Time Java Platform Programming*, Prentice Hall, 2002.

Scheduling Policy Costs on a JAVA Microcontroller

Leomar S. Rosa Jr., Flávio R. Wagner, Luigi Carro,
Alexandre S. Carissimi, and André I. Reis

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
PO Box 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{leomarjr, flavio, carro, asc, andreis}@inf.ufrgs.br

Abstract. This paper presents the implementation of different scheduling policies on a Java microcontroller. Seven new instructions were added to the architecture to support context switching and scheduler implementation. By using these instructions, four schedulers following the POSIX standard were developed for the specific architecture. These schedulers were used in a study about the impact of different scheduling policies for embedded systems applications. Several design costs are discussed, including the hardware cost of the extended instructions, ROM and RAM capacity used, the number of cycles to run the chosen scheduler and the application, and also the power consumption overhead. Experiments show that the exploration of different scheduling alternatives as well as careful scheduler implementation may play an important role in performance optimization.

1 Introduction

Embedded systems are fundamental parts of modern life appliances. These equipments perform several different tasks, using a limited set of resources. The complexity of applications and underlying hardware, tight performance/power budgets, as well as an aggressive time-to-market design schedule require the use of run-time software support by application developers. This support usually takes the form of an operating system [12] [13] that manages four system aspects: file systems, memory access management, I/O system, and processor use. The processor use is managed by a routine of the operating system called scheduler. A scheduler is needed when a single processor must handle different tasks. The scheduler may be implemented in software [11] or in hardware [8].

The power consumption of a complete embedded operating system was investigated in [1] [2] [3] [15]. The work of [2] investigates the power consumption of the PalmOS operating system. The power consumption of the μ C/OS running on a Fujitsu SPARCLite platform is analyzed in [3]. The energy consumption of eCos, focusing in particular on the relationship between energy consumption and processor frequency was characterized in [1]. The work of [15] compared the consumption of the Linux OS running on a StrongARM-based platform and μ C/OS running on a Fujitsu SPARCLite platform. The work [16] investigates the performance and silicon area requirements of a multithreaded Java microcontroller. However, none of these works studied the influence of different scheduling policies in terms of area, power

consumption, and performance together. In this paper we investigate the implementation cost for different scheduling policies implemented in software. Therefore, the operating system support for file systems and memory management is neither considered nor implemented. There are many applications in which there is no need for this kind of support, and these aspects will be studied in a future work. Our study will concentrate on a dedicated leveraged OS consisting only of schedule support.

The goal of this paper is to investigate the cost of implementing software schedulers for embedded applications, in terms of area, performance, and power overhead. All experiments have been carried on top of a Java microcontroller [5] [6].

This paper is organized as follows. Section 2 details the processor architecture. New instructions to support scheduler implementation and context switching are presented in Section 3. The implemented scheduling policies are discussed in Section 4. Results and conclusions are presented in Sections 5 and 6, respectively.

2 The FemtoJava Microcontroller

The FemtoJava Microcontroller [5] [6] is a stack-based microcontroller that executes Java bytecodes, whose major characteristics are: reduced bytecode instructions set, Harvard architecture, orthogonality of execution, small size, and easy insertion and removing of instructions depending on the target application.

FemtoJava implements in hardware an execution engine for Java, through a stack machine compatible with the Java Virtual Machine (JVM) specification, like PicoJava [10]. In general, the JVM has three major components: class loader, class verifier, and execution engine. In fact, the class loader and verifier act at runtime and are only necessary if one wants a multi-application platform and has to download code over a network. A compiler that follows the JVM specification is being used and will synthesize an ASIP version of FemtoJava. Only the execution core and some tools to extract the software at design time are really necessary.

An immediate advantage of native execution of Java bytecodes is the software compatibility. This feature guarantees the availability of cross-platform development of software. In some conventional Java platforms (for example, PCs or workstations) one can just implement and run a Java program. Running the program is equivalent to simulating the behavior of the application in the target microcontroller, with all resources and convenience of a desktop environment in the development phase.

The Java Virtual Machine is an abstract machine, based on a stack architecture, with Java bytecode execution capability [9]. Small applications as smart cards, building access control, and active badge location are not a good target for powerful superscalar processors, with large caches and register files running a full Java execution environment. For this kind of application just a small, low cost and cacheless microcontroller could be an optimal solution. In addition, the stack machine has characteristics and limitations distinct from those of CISC and RISC ones. Zero-operand stack machines do not encode operand information on the instruction word. Therefore, stack code is more compact and portable, because it needs fewer bits to encode one instruction, and makes no assumption about register file organization (making the instruction size small) [7].

Table 1. Description of the instructions for scheduling support

Instruction	#of μInstructions	Bytecode	Example	Meaning
INIT_VAL	9	f4	f4 \$s1,\$s2	Mem[\$s2] ← \$s1
INIT_STK	9	f5	f5 \$s1,\$s2	Mem[\$s2] ← \$s1 - 2
REST_CTX	11	f6	f6 \$s1	SP ← Mem[\$s1]
SAVE_CTX	7	f7	f7 \$s1	Mem[\$s1] ← SP
SCHED_THR	12	f8	f8 \$s1,\$s2	if (Mem[\$s1]=0) PC ← PC + \$s2 ; else PC++
SCHED_PRT	11	f9	f9 \$s1,\$s2	A ← \$s1 , B ← \$s2
GET_PC	7	fa	fa \$s1	PC ← Mem[\$s1]

from the JVM that is a stack machine. Seven new instructions have been created for this, adding to those already existing in the architecture. Table 1 presents these new instructions. The number of microinstructions varies from 7 to 12. Each of the instructions is described below.

INIT_VAL. This instruction saves the value of a register (s1) in a memory position pointed by another register (s2). It is used to save information needed to schedule the processes.

INIT_STK. This instruction is needed to initialize a stack for each process to be treated by the scheduler. As the push instruction pre-increments the Stack Pointer (SP) and the memory is addressed by byte, it is necessary to decrement the saved address by two positions.

REST_CTX. This instruction restores the SP from the memory position pointed by register s1. As the FemtoJava microcontroller saves the context in the stack before handling interruptions, the remaining context is recovered automatically from the stack.

SAVE_CTX. This instruction saves the SP in the memory position pointed by s1. As the FemtoJava microcontroller saves the context in the stack before handling interruptions, the remaining context is saved automatically in the stack.

SCHED_THR. This instruction is used to redirect the execution flow to the process that is granted with CPU access by the scheduler.

SCHED_PRT. It transfers the priority values to ALU registers A and B and compare them by a subsequent arithmetic instruction. This instruction is used to compare the priorities of two different processes and determine which one has the higher priority. This instruction is used only by schedulers considering processes with different priorities.

GET_PC. This instruction is an unconditional branch used to jump to a memory position pointed by register s1. It is used to jump to some specific sub-routines of the scheduler.

The hardware overhead for implementing the new instructions with Altera MaxplusII on a FLEX10K EPF10K70RC240-2 device is shown in Table 2. The number of logic cells (LC) of the original FemtoJava processor without the new instructions is 2057 logic cells. This number is increased to 2173 logic cells when the six instructions for scheduling without priority (f4, f5, f6, f7, f8, fa) are implemented. The number of cells to implement all the new instructions, including instruction

Table 2. Hardware overhead and number of logic cells (LC) for implementing the new instructions with Altera MaxplusII on a FLEX10K EPF10K70RC240-2 device

Schedulers	# of New Instructions	LC	% Area
None	0	2057	100%
Without Priority	6 (f4, f5, f6, f7, f8, fa)	2173	105.6 %
With Priority	7 (f4, f5, f6, f7, f8, f9, fa)	2175	105.7 %

SCHED_PRT that is used only for schedulers considering processes with different priorities, is 2175. The area overhead for extending the FemtoJava instruction set to support context switching and allow the implementation of a software scheduler using the new instructions is of the order of 120 logic cells and represents an area increase of around 5% with respect to the original microcontroller.

4 Implemented Schedulers

This section discusses the implemented scheduling policies. An operating system must allocate computer resources among the potentially competing requirements of multiple processes. In the case of the processor, the resource to be allocated is execution time on the processor and the means of allocation is scheduling. This way, the scheduler is the component of the operating system responsible to grant the right to CPU access to a list of several processes ready to execute. This idea is illustrated in the five-state diagram of Fig. 2 [14]. The meaning of each state is defined as follows:

New: a new process may be created and admitted to the list of ready processes.

Ready: this is a list of processes that are ready to execute. Once the processor is free, the scheduler must choose one of them to become active in the CPU. This is called dispatch. The choice of the process is done following a scheduling policy.

Running: it is a process that is being run on the processor. There are three ways a process may loose processor access. The scheduler may time-out the process, the process may request an external operation or execute a synchronization primitive and become blocked until the request is granted, or the process may finish and be released for exit.

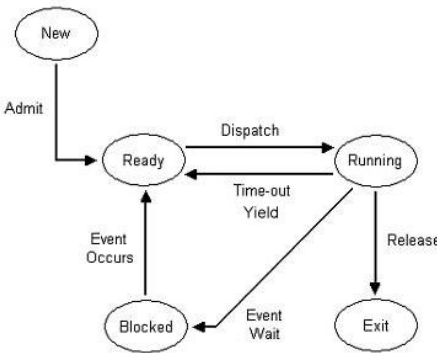


Fig. 2. Five-State process model

Blocked: contains a list of processes waiting for pending events (I/O or sincronization).

Exit: finished processes.

Four schedulers following the POSIX standard [4] for operating systems [14] were implemented: FIFO and Round-Robin with and without support to weighted process priority. The implemented schedulers are discussed in the following sub-sections.

4.1 FIFO

The FIFO scheduling policy just dispatches the ready processes in a first-in-first-out basis. Each process is executed until it finishes or until it requests an external operation. In the case of an external request, the process will be blocked to wait for an event and, once the request is granted, it will be inserted in the end of the list of ready processes. The implementation of this scheduling policy is straightforward and requires mainly context switching, performed by saving and restoring special registers like the Stack Pointer and the Program Counter. This policy was implemented directly in bytecodes by using the new instructions.

4.2 FIFO with Priority

The FIFO with priority scheduling is very similar to the FIFO policy. Each process has an associated priority, and the scheduler just dispatches the ready process with the highest priority. This way it visits all the processes in the ready state and dispatches the one with the highest priority. If there are several processes with the same priority, the FIFO policy is adopted among these processes. The implemented FIFO is non-preemptive.

4.3 Round-Robin

The Round-Robin scheduling policy works with the concept of a time quantum. Each process receives a time quantum to use the processor. Each process may be deactivated due to three reasons: granted time ends, process finishes, or process requests external event. The concept of time quantum is implemented through a timer interrupt programmed by the scheduler. The scheduler sets a timer to count the number of cycles corresponding to the quantum and dispatches a process. When the timer achieves the quantum count, it produces a timer interrupt that transfers the processor control back to the scheduler that will dispatch the next process in the list of ready processes. In the implemented scheduler, the user may configure the time quantum. In Section 5, four different time quanta were used to measure the scheduling costs.

4.4 Round-Robin with Priority

The Round-Robin with priority scheduling is very similar to the Round-Robin policy. Each process has an associated priority, and the scheduler adopts a Round-Robin

Table 3. Memory overhead introduced by the schedulers

Scheduler	RAM overhead due to internal data	ROM overhead due to scheduler code
FIFO	41 bytes	1567 bytes
FIFO with Priority	42 bytes	1111 bytes
RR	41 bytes	1596 bytes
RR with Priority	42 bytes	1161 bytes

policy among the ready processes with the highest priority. The implemented scheduler is non-preemptive.

4.5 Memory Overhead

The memory overhead introduced by the schedulers code is presented in Table 3. It is possible to notice that there is a small RAM overhead due to some internal scheduler data. The size of the code of each scheduler introduces a ROM overhead that is quite small when compared to the code of usual applications.

4.6 Implementation of the Schedulers

Some implementation details are presented in the appendixes and illustrate the use of the new Femtojava instructions. Appendix I details the implementation of process initialization, while Appendix II describes the code for process scheduling.

5 Results

Four different schedulers following the POSIX standard [4] were implemented. As described previously, these schedulers introduce a small hardware overhead due to the new instructions and a small memory overhead due to the scheduler code itself (ROM overhead) and to some internal data (RAM overhead). In this section we discuss the relative performance of the different schedulers considering the number of cycles they use to complete a set of tasks, as well as the power consumption involved. First of all, let us consider the different schedulers in use. The Round-Robin schedulers are characterized by a time quantum, and the time quanta used in our experiment are shown in Table 4. The schedulers considering different priorities may treat different sets of tasks, with different priorities. Table 5 shows the priority of each task used in our experiments, as they are ordered in the program ROM. Higher number means higher priority. Each task is a sorting algorithm that processes a 10-element vector. The set S1 has ten tasks of equal priority. The sets S2 to S4 have different priorities, but the position the tasks are placed in the instruction ROM are different: set S2 has tasks ordered in increasing priority, set S3 has tasks ordered in decreasing priority and set S4 has tasks in random order.

The number of cycles and power consumption needed to compute the set of 10 sorting tasks is shown in Table 6 and Table 7. Table 7 shows the estimated power

Table 4. Time quanta used for Round-Robin schedulers

Time Quantum	# of cycles
Q1	7.000 cycles
Q2	5.000 cycles
Q3	3.000 cycles
Q4	2.500 cycles

Table 5. Process priorities used in the schedulers considering processes with different priorities

Mode	Process Priority
S1	[5] [5] [5] [5] [5] [5] [5] [5] [5] [5]
S2	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
S3	[9] [8] [7] [6] [5] [4] [3] [2] [1] [0]
S4	[7] [6] [8] [2] [1] [7] [5] [3] [4] [9]

Table 6. Number of cycles for different schedulers on the same tasks

Scheduler	Quantum	Priority	Cycles
FIFO	-	-	66.514
FIFO with Priority	-	S1	66.797
		S2	72.953
		S3	66.797
		S4	68.207
RR	Q1	-	66.517
	Q2		73.069
	Q3		76.328
	Q4		82.880
RR with Priority	Q1	S1	66.800
	Q2		73.207
	Q3		76.432
	Q4		82.839
	Q1	S2	72.956
	Q2		88.638
	Q3		95.118
	Q4		109.085
	Q1	S3	66.800
	Q2		73.207
	Q3		76.432
	Q4		82.839
	Q1	S4	68.210
	Q2		76.003
	Q3		79.837
	Q4		87.588

consumption produced by the schedulers in number of switched gate capacitances (SGC). This power consumption was obtained by simulation. The simulator used to perform this estimation is a high level simulator that estimates the number of gate

Table 7. Power consumption for different schedulers on the same tasks

Scheduler	Q.	P.	RAM	ROM	Core	Total
FIFO	-	-	223.606.000	4.378.510	119.062.403	347.046.913
FIFO		S1	223.169.000	4.397.830	119.738.075	347.304.905
FIFO with Priority	-	S2	243.225.000	4.901.300	130.453.446	378.579.746
		S3	223.169.000	4.397.830	119.750.409	347.317.239
		S4	227.700.000	4.512.140	122.212.857	354.424.997
	Q1		224.342.000	4.379.200	119.202.709	347.923.909
RR	Q2	-	243.064.000	4.874.390	130.823.661	378.762.051
	Q3		252.379.000	5.121.180	136.601.747	394.101.927
	Q4		271.101.000	5.616.370	148.243.343	424.960.713
	Q1		223.905.000	4.398.520	119.878.654	348.182.174
	Q2	S1	241.477.000	4.886.810	131.398.505	377.762.315
	Q3		250.332.000	5.133.830	137.205.995	392.671.825
	Q4		267.904.000	5.622.120	148.749.841	422.275.961
	Q1		243.961.000	4.901.990	130.592.209	379.455.199
	Q2	S2	291.778.000	6.152.960	158.220.834	456.151.794
RR with Priority	Q3		311.443.000	6.666.780	169.694.899	487.804.679
	Q4		353.855.000	7.776.070	194.361.976	555.993.046
	Q1		223.905.000	4.398.520	119.890.781	348.194.301
	Q2	S3	241.477.000	4.886.810	131.421.625	377.785.435
	Q3		250.332.000	5.133.830	137.238.112	392.703.942
	Q4		267.904.000	5.622.120	148.799.574	422.325.694
	Q1		228.436.000	4.512.830	122.352.506	355.301.336
	Q2	S4	250.516.000	5.115.200	136.301.339	391.932.539
	Q3		261.441.000	5.411.900	143.175.839	410.028.739
	Q4		283.429.000	6.010.590	157.072.502	446.512.092

capacitances being switched for every processor microinstruction, as described in [17].

According to data in Table 6, it is possible to observe that the FIFO scheduling policy is always the faster one. The execution time penalty of the Round-Robin schedulers increases with the decreasing of the size of the time quantum. However, a smaller time quantum will give a better service quality, because the processes will execute more frequently. This is an important point to be considered in RTOS. The FIFO or a Round-Robin with large time quantum will consume less power, but the quality of service may be degraded due to the excessive time that a process may have to wait to have the CPU granted.

Notice that in the implementation of the schedulers dealing with different task priorities, the scheduler saves some information of the process with highest priority, and this way the order of the processes in the program ROM will have a significant impact on the performance and power consumption. This may be observed in Tables 6 and 7 by comparing the Round-Robin with priority and quantum Q4 running for sets S2 and S3. Set S2 has an unfavorable order, and the number of cycles is around 109k cycles. Set S3 has a favorable order, and the number of cycles is around 83k cycles. This represents a difference of 30% due to a better scheduler implementation. The power consumption overhead follows a similar pattern and it is shown in Table 7. This way, the exploration of different scheduling alternatives as well as careful scheduler implementation may play an important role in performance and power consumption optimization.

6 Conclusions and Future Work

Four different schedulers following the POSIX standard [4] were implemented. As described previously, these schedulers introduce a small hardware overhead due to the new instructions added to the FemtoJava architecture and a small memory overhead due to the scheduler code itself (ROM overhead) and to some internal data (RAM overhead). Simulations of the schedulers indicate that the exploration of different scheduling alternatives as well as careful scheduler implementation may play an important role in performance and power optimization.

Future work will analyze the impact of other scheduling policies. This analysis will be used to create an automatic tool to synthesize embedded schedulers according to particular system requirements. This linker tool is being developed and will allow the user to configure one of the implemented schedulers following a given policy to be linked and manage several applications sharing the same FemtoJava CPU.

References

1. Acquaviva A.; Benini L.; Ricco B. Energy Characterization of Embedded Real-Time Operating Systems. In Proc. Workshop Compilers & Operating Systems for Low Power. (2001)
2. T.L.Cignetti, K.Komarov, C.S.Ellis. Energy Estimation Tools for the Palm. In: Proceedings of the ACM MSWWiM (2000)
3. Dick, R.P.; Lakshminarayana, G.; Raghunathan, A.; Jha, N.K. Power Analysis of Embedded Operating Systems, 37th Design Automation Conference, pp. 312–315. (2000)
4. Gallmeister, B. O. POSIX.4 Programming for the Real World. First Edition. O'Reilly & Associates, Inc. (1995)
5. Ito, S. et al. System Design Based on Single Language and Single-Chip Java ASIP Microcontroller, In Proceedings of Design Automation and Test in Europe, pp. 703–707, Paris, France. IEEE Computer Society Press. (2002)
6. Ito, S. A., Carro, L. and Jacobi, R. Making Java Work for Microcontroller Applications, IEEE Design & Test, vol. 18, no. 5, pp. 100–110, Sep–Oct. (2001)
7. Koopman, P. Why stack machines? In the URL: <http://www.cs.cmu.edu/koopman/forth/whystack.html>. (2002)
8. Kreuzinger, J.; Brinkschulte, U.; Pfeffer, M.; Uhrig, S.; Ungerer, Th. Real-time Event-Handling and Scheduling on a Multithreaded Java Microcontroller, Microprocessors and Microsystems, vol. 27, pp. 19–31. (2003)
9. Lindholm, T. and Yellin, F. The Java Virtual Machine Specification. The Java Series, Addison-Wesley. (1997)
10. Mcghan, H. and O'Connor, M. Picojava: A Direct Execution Engine For Java Bytecode, Computer, vol. 31, no.10, pp. 22–30, Oct. (1998)
11. Memik, S. O.; Bozorgzadeh, E.; Kastner, R.; Sarrafzadeh, M. A Super-Scheduler for Embedded Reconfigurable Systems, IEEE/ACM International Conference on Computer Aided Design, pp. 391–394. (2001)
12. Ortiz, S. Jr. Embedded OSs Gain the Inside Track. IEEE Computer, vol. 34, n. 11, pp. 14–16. (2001)
13. Schlett, M. Trends in Embedded-Microprocessor Design, IEEE Computer, vol. 31, n. 8, pp.. 44–49. 1998
14. Silberschatz, A.; Galvin, P.; Gagne, G. Applied Operating System Concepts. First Edition. Wiley. (2000)

15. Tan, T.K.; Raghunathan, A.; Jha, N.K. Embedded Operating System Energy Analysis and Macro-Modeling, IEEE International Conference on VLSI in Computers and Processors, pp. 515–522. (2002)
16. Kreuzinger, J.; Zulauf, R.; Schulz, A.; Ungerer, T.; Pfeffer, M.; Brinkschulte, U.; Krakowski, C. Performance Evaluations and Chip-Space Requirements of a Multithreaded Java Microcontroller. To be published. Available document in Scientific Literature Digital Library, CiteSeer: <http://citeseer.nj.nec.com/384138.html>
17. Beck Filho, A. C. S.; Mattos, J. C. B.; Wagner, F. R.; Carro, L.. CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. In 16th Symposium on Integrated Circuits and Systems Design Proceedings, IEEE Computer Society Press. (2003)

Appendix I: Initialization Module for the Round-Robin Scheduler

The instructions with address ranging from 2dH to 37H implement the initialization of the timer and of the interrupt system for the schedulers based on fixed time quanta. The code starting at the address 38H implement the stack initialization for each process, as well as the initialization of some information present in the process table (stack pointer, process priority, process status). This piece of code uses the new instructions INIT_STK and INIT_VAL.

```

2d : 10; -- bipush -- ENABLE INT
2e : 22; --
2f : 10; -- bipush
30 : 00; --
31 : f2; -- store_idx
32 : 11; -- sipush -- SET TIMER
33 : 00; --
34 : 14; --
35 : 10; -- bipush
36 : 0c; --
37 : f2; -- store_idx
38 : f5; -- INIT_STK
39 : e6; --
3a : 7f; --
3b : 05; -- PC 2 value initialized in e67dH
3c : 1d; -- PC 2 value initialized in e67dH
3d : f4; -- INIT_VAL
3e : 00; --
3f : f2; --
40 : e6; -- SP 2 initialized in f2H
41 : 7a; -- SP 2 initialized in f2H
42 : f5; -- INIT_STK
43 : cc; --
44 : ff; --
45 : 05; -- PC 3 value initialized in ccfdH
46 : 84; -- PC 3 value initialized in ccfdH
47 : f4; -- INIT_VAL
48 : 00; --
49 : f3; --
4a : cc; -- SP 3 value initialized in f3H
4b : fa; -- SP 3 value initialized in f3H

```



```

.
.
92 : f4; -- INIT_VAL (ffff = ready , 0000 = blocked)
93 : 01; --
94 : 01; --
95 : ff; -- state of process 1 saved in 0101H
96 : ff; -- state of process 1 salvo in 0101H
97 : f4; -- INIT_VAL
98 : 01; --
99 : 02; --
9a : ff; -- state of process 2 saved in 0102H
9b : ff; -- state of process 2 saved in 0102H
.
.
c4 : f4; -- INIT_VAL
c5 : 01; --
c6 : 11; --
c7 : 00; -- priority of PROC 1 saved in 0111H
c8 : 05; -- priority of PROC 1 saved in 0111H
c9 : f4; -- INIT_VAL
ca : 01; --
cb : 12; --
cc : 00; -- priority of PROC 2 saved in 0112H
cd : 05; -- priority of PROC 2 saved in 0112H

```

Appendix II: Task Scheduling for the Round-Robin Scheduler

The following piece of code illustrates the use of the new instructions for task scheduling. The instructions SCHED_THR, SAVE_CTX, SCHED_PRT and REST_CTX are used. The first actions taken by the code is to disable the timer (addresses 133 to 137) and then save the context of the current process (addresses 138 to 16b). The goal of this piece of code is to guarantee that any other interruption will not be handled during the execution of the scheduler code, and also to guarantee the integrity of the data belonging to the process being removed from the CPU. The scheduling policy code begins at the address 199H.. This code chooses the next process to be scheduled. The context restoring for the process gaining access to CPU starts at address 471H. Finally, the timer is enabled, with the code starting at address 4aaH. This action is performed before transferring the CPU control to the process being scheduled. It is important to notice that this piece of code will vary according to the scheduling policy being used. The code shown here implements a Round-Robin policy.

```

133 : 10; -- bipush -- STOP TIMER
134 : 00; --
135 : 10; -- bipush
136 : 0d; --
137 : f2; -- store_idx
138 : f8; -- SCHED_THR
139 : 01; --
13a : 21; --

```

```

13b : 25; --
13c : f8; -- SCHED_THR
13d : 01; --
13e : 22; --
13f : 27; --
.
.
160 : f7; -- SAVE_CTX PROC 1
161 : 00; --
162 : f1; --
163 : a7; --
164 : 00; --
165 : 34; --
166 : f7; -- SAVE_CTX PROC 2
167 : 00; --
168 : f2; --
169 : a7; --
16a : 00; --
16b : 2e; --
.
.
199 : f4; -- INIT_VAL
19a : 01; --
19b : 1f; --
19c : 00; --
19d : 00; --
19e : f8; -- SCHED_THR
19f : 01; --
1a0 : 01; --
1a1 : 45; --
1a2 : f9; -- SCHED_PRT
1a3 : 01; --
1a4 : 11; --
1a5 : 01; --
1a6 : 1f; --
1a7 : a2; -- IF_ICMPGE
1a8 : 00; --
1a9 : 3d; --
.
.
471 : f6; -- REST_CTX PROC 1
472 : 00; --
473 : f1; --
474 : a7; -- GOTO SET_TIMER, START_TIMER, RETURN
475 : 00; --
476 : 34; --
477 : f6; -- REST_CTX PROC 2
478 : 00; --
479 : f2; --
47a : a7; -- GOTO SET_TIMER, START_TIMER, RETURN
47b : 00; --
47c : 2e; --
.

```

```
.  
4aa : 10; -- bipush -- SET TIMER  
4ab : 14; --  
4ac : 10; -- bipush  
4ad : 0c; --  
4ae : f2; -- store_idx  
4af : 10; -- bipush -- START TIMER  
4b0 : 03; --  
4b1 : 10; -- bipush  
4b2 : 0d; --  
4b3 : f2; -- store_idx
```

Experiences and Benchmarking with JTime

Greg Bollella¹, Krystal Loh², Graham McKendry¹, and Thomas Wozenilek²

¹ Sun Microsystems Laboratories Europe
180 Avenue de l'Europe, ZIRST de Montbonnot,
38334, SAINT ISMIER Cedex, France

² Sun Microsystems Laboratories
2600 Casey Avenue, MTV29,
Mountain View, CA 94043, U.S.A.

{greg.bollella,krystal.loh,graham.mckendry,thomas.wozenlek}
@sun.com

Abstract. JTime is a commercial offering from TimeSys, Inc. implementing the Real-Time Specification for Java (RTSJ). We have been experimenting with and benchmarking JTime for a number of months and report some precise time measurements of the predictability, repeatability, and accuracy of JTime's implementation of some key RTSJ constructs. In particular, we have measured the interarrival times between subsequent releases of the `waitForNextPeriod()` method call (in three different thread contexts) and for instances of `PeriodicTimer`. We also measured latencies for the above constructs and for instances of `AsyncEventHandler`. We find, in general, that the results are very good and that applications with hard real-time constraints can be accommodated. We further note that these results are obtained using the interpreted version of JTime and would expect significantly better results with pre-compilation.

1 Introduction

For brevity, we omit introductory discussions of the motivation for using the Java Language in real-time and embedded systems and about the development of the Real-Time Specification for Java (RTSJ). We are fairly certain that attendees of this workshop have sufficient background, however, information on the RTSJ can be found at [1], [7] and a comprehensive criticism at [3].

With respect to predictability measurements of RTSJ implementation there is very little related work so far. Two efforts have been published: Sharp, et al, looked at early versions of the TimeSys RTSJ implementation in the context of military aircraft avionics management systems [4], and Corsaro, et al, examined performance of their RTSJ compiler [2].

1.1 Goals

The ultimate goal of this work is to provide a comprehensive, precise, and accurate timing analysis of the TimeSys implementation of the RTSJ, a brief description of the

experience we acquired, and difficulties we overcame in obtaining sufficient data for such an analysis. At this point, we are some way into our analysis and report on early findings, but in a very real sense we have just begun. We expect to continue this work over the next year and to more fully develop our analysis and reports.

Section 2 overviews our benchmarking plan with respect to RTSJ constructs. Our goal here is to create, in the literature, an archive of analysis test templates which developers can apply to implementations of the RTSJ to better understand which RTSJ constructs are suitable for particular requirements of their applications and which are not. Note that although we use JTime as the test platform, we have attempted to design the tests to expose the strengths and weaknesses of the RTSJ constructs themselves, not just the test platform. We expect to make our tests freely available for others to verify our results and, in the future, analyze various other implementations of the RTSJ.¹

Early in the effort, we were surprised with the difficulty in determining the precise clock frequency of the processor of our test platform. We had to design a systematic test suite to bracket the frequency after realizing that a simple test was unlikely to produce a value with the precision we required. Details of the test suite appear in Section 3.

We next detail the results of our experiments ordered by RTSJ construct: first, periodic processes implemented by threads in Section 4, then periodic timers in Section 5, and finally asynchronous events and handlers in Section 6. In each of these three sections we describe our procedure, state our performance criteria, expose our results, and offer an analysis.

The RTSJ claims to support hard real-time constraints and defines hard real-time to be the ability of the system to guarantee that feasible schedules of tasks will all meet their deadlines. In Section 7 we describe a test which shows that this implementation of the RTSJ may indeed be able to make such a guarantee.

Section 8 concludes with our observation that the results so far should be considered astoundingly good. The predictability and low latencies achieved are unprecedented in the real-time Java community.

2 Benchmarking Overview

2.1 Characterizing Temporal Performance

The RTSJ embraces the abstract notion that correct temporal performance depends primarily on the temporal predictability of the execution of logic *as defined by* the application and executed by the underlying implementation. Thus, when the RTSJ application indicates that a particular piece of logic must become available for execution at a particular time, it is the quality of the RTSJ implementation which determines how well this requirement is met. We designed our tests to understand

¹ To this end, Caltech's Jet Propulsion Laboratory is establishing an RTSJ performance test repository and is actively seeking additional tests and a volunteer organization to host, maintain, and manage the tests.

how well a particular implementation meets this requirement and to allow comparison between different RTSJ constructs which might accomplish equivalent semantics.

Our predictability tests are classified broadly into a four cell matrix. The two rows are: 1. tests which measure the variability in interarrival times for a particular piece of logic (task) and, 2. tests which measure the latency between a logical point in time, e.g., the theoretical release time of a task, and a real point in time, e.g., the time a task begins to execute. The two columns are: 1. a quiescent system and, 2. with auxiliary tasks aggressively producing garbage.²

Our predictability tests attempt to expose two characteristics of the implementation, overhead and preemptibility (until, however, we directly measure utilization we will not be able to separate overhead and preemptibility and report them logically as a single concept). By overhead we mean any virtual machine (JVM) or operating system (OS) activity, e.g., dispatch decisions necessary for managing the tasks, which causes variability in our measurements and by preemptibility we mean the similar interference of non-preemptible sections, e.g., locks, of the JVM and OS.

Both predictability test types require a carefully controlled test environment so that we may confidently assert that our measurements truly measure the overhead and non-preemptible sections we intend to quantify and not other characteristics or activities. To this end, each measurement section will describe how we controlled the system. For example, an interarrival measurement of a task will not reflect overhead and preemptibility if other tasks of equal or higher priority are executing concurrently so we take all possible precautions to ensure that no unknown task interferes with our measurements.

2.2 Measurement Calculations

For each test we measure either interarrival times, latencies, or both. Release times are denoted as r_i , $0 \leq i \leq n$ for n periods or test iterations. The time at which the i^{th} invocation of logic, corresponding to the i^{th} release, begins to execute is denoted as e_i , $0 \leq i \leq n$.

The interarrival value is the time between the execution start times for subsequent releases, $I_i = e_{i+1} - e_i$.

Latency is the value measured between the release time and the start of execution, $L_i = e_i - r_i$. We define jitter as the difference between the longest and the shortest latencies and compute jitter in two ways, using both the interarrival and latency measurement streams as raw data: $J_{int} = ((\max(I_i)) - \min(I_i))/2$, ($0 \leq i \leq n$), $J_{lat} = \max(L_i) - \min(L_i)$, ($0 \leq i \leq n$). If the process under test occurs every period J_{int} should be approximately the same as J_{lat} but if latencies cause deadline misses the latency jitter will increase whereas the interarrival jitter could remain small. Computing jitter in these two ways give us better insight into the behavior of the RTSJ and underlying OS.

² As this work progresses we plan to measure utilization and average throughput directly.

2.3 Hardware and Software

This section details characteristics of the hardware and software platforms used in our testing. The system platform is based on an Intel Celeron 1.10 GHz CPU with 128MB SDRAM running with a 100 MHz front-side data bus. The operating system is Red Hat Linux 8.0 running the TimeSys Linux/NET for x86 kernel, build 4.0.14. The Java compiler came from the Sun JDK 1.4.1_01. The real-time VM and libraries came from JTime 1.0-446.1-DBG. The VM options for memory space sizes are consistently set for a heap size of 64MB, a thread stack size of 4MB, and an immortal memory area of size 6MB.

Before executing the VM for each test set, all kernel level threads were set to a priority level of 9, which is 1 less than the maximum user priority. This is accomplished using the command, `rtprio`, a utility packaged with JTime that allows the user to specify the priorities of all system processes, including interrupt request lines and handlers. As well, the VM is given a scheduling priority of `-20` using the UNIX command-line command, `nice`. All non-essential user processes are killed before VM execution, leaving only 3 necessary user processes: `init`, `login` and `bash`.

We here note an important decision reflected in our results:

If we do not terminate all non-essential user processes, all measurements are better, i.e., less jitter and smaller latencies, by about a factor of five. We choose to report the worse results because we are attempting to analyze worst-case behavior. Although it is likely that a particular use of the TimeSys implementation would not terminate these processes it is possible that one could inadvertently fail, causing these higher latencies and jitter to occur and to compromise the application had the worst-case not been included in the analysis.

3 Determining Clock Frequency

We, of course, want our measurements to have as fine a resolution as possible. However, we note that the resolution of a time value data field defined by an OS or JVM, e.g., nanoseconds, is unrelated to, what we term, the clock update rate. Most time values offered by OSs are ‘soft clocks’, i.e., they maintain their current value in a software defined data field. For a clock with nanosecond resolution it is obvious that an OS would not likely *update* the data field every nanosecond, but possibly ever few microseconds. This difference between perceived clock resolution and the update resolution may cause anomalous results in timing tests unless it is detected and corrected in computations. It will, even in the corrected case, cause timing results to be less precise than possibly perceived.

Fortunately, the processor of our test hardware platform has a 64 bit hardware register which is incremented once for each cycle of the processor clock. The register has, at all times, a value equal to the number of processor cycles that have occurred since the processor was powered on. We use values from this register to measure the interarrival periods during our interarrival type tests.

However, knowing, with high precision, how many cycles have occurred in a given interval is useful in determining variation but non-intuitive for the human reader. We would like the reader to be able to interpret the results in time units and thus we require the frequency to be able to convert the cycle counts to time units. This section details our effort at determining the actual frequency of the processor clock on our hardware test platform so that we can use the cycle count hardware register to measure, at least, interarrival times, but then report our findings in time units. As a note, for interarrival measurements, we do all of our computations using cycles and convert to time only once for display. It is unfortunate, but even with the most precise determination of clock frequency it is not possible to obtain accurate results using the hardware register for *latency measurements* because the error induced in the measurements by the error in the clock frequency would be cumulative. Thus, for latency measurements we use a softclock and report our results with appropriate error bounds.

The manufacturer reports 1.10 GHz as the clock speed of the test platform processor. The precision of this value would then limit our results to an accuracy of ± 5.0 ms, far too large for the kind of tests we envision. Our first attempt to determine a more precise value was to check the clock frequency stored by Linux at boot time in the `proc` directory. This value, because we observed significant variation between boots was not precise enough.

Our algorithm for determining clock frequency is: get and store the cycle count and the system time (using `clock_gettime()`), call `sleep()` for a variable amount of time, then get and store the count and time again. Over a period of weeks, periodically running this test we arrived at an average of 1,102,516,089 Hz with a standard deviation of 4,687 Hz.

However, there are interesting observations to report not evident in the average and standard deviation. On a single day the maximum and minimum frequency recorded varied by *no more* than 10,000 Hz. Between days, but on a slowly oscillating waveform, we could see a maximum and minimum frequency that varied no more than 30,000 Hz.

Our analysis of these observations is that the update rate of `clock_gettime()` is in the 10 μ s range and that some other unknown factor caused the day-to-day variation. We suspect this unknown factor to be temperature, affecting the crystal oscillation frequency, but we do not have the equipment to verify this assumption.

We can, however, assert with confidence that for the runs of a particular day our final reported results, for both interarrival and latency measurements, are accurate to ± 5.0 μ s.

4 Periodic Threads

Both new thread types introduced by the RTSJ can be used for configuring and handling periodic events. Scheduling parameters for a periodic thread, such as the thread priority, are specified at creation time. Release parameters, set the start time, period length, cost and deadline of the periodic thread are also defined at the time the thread object is created. The method `waitForNextPeriod()` is used to block the periodic

thread until the start of the next period. A normal Thread can approximate *unreliable* periodic work simply by calling `sleep()` for the remaining period length.

4.1 Procedure

We create periodic processes using three thread classes, Thread, RealtimeThread (RT), and NoHeapRealtimeThread (NHRT). All three processes are tested both with the system as quiescent as possible and with application threads producing a significant number of unreferenced objects (garbage). We refer to these system states as quiescent and GP. NHRT is also tested with a quiescent system for the first half of the run and GP system for the remainder of the run.

The general setup for the periodic process test involves first creating and starting GP threads, if appropriate. This is followed by creating and starting an RT with a maximum real-time priority that enters immortal memory. From within immortal memory, the periodic test thread is created and started. The initial RT is put to sleep until the allotted test time has passed. Upon waking, the RT signals the periodic test thread to stop executing. Finally, any GP threads are stopped and there is then a transfer of collected runtime data from memory to disk.

Various settings are held constant for each test run. The relative time when the first period begins is called the periodic start time and is set to 10 s. The period is the duration of each interval and is set to 50 ms, which is known to be large enough for the thread to complete timing measurements well before the start of the next period. Cost refers to the processing time required per period. The value is set to 1 ms, but is not actually used by the system. The deadline is the latest possible completion time relative to the start of each period and is set to 50 ms. The thread logic is quite simple and involves the tasks of querying and storing the system cycle count and high resolution time, as well as checking for end of the array index used for storing the time values.

The priority of the test thread has a default value of `PriorityScheduler.MAX_PRIORITY` less 5. The miss handler priority is one less. The priority of the RT thread that creates, starts and stops the periodic test thread has a value of `PriorityScheduler.MAX_PRIORITY`. The priority of the garbage producing threads is `Thread.MIN_PRIORITY`.

Runtime data is collected on different aspects of the periodic threads. The running time is the approximate amount of time from when the test thread is started to when it completes its last period. It has a typical value of 10 min. The total number of Integer objects created by GP threads is the GP object count. The data set containing the system cycle counts of the start of execution for each period is called periodic cycles. Specifically, the cycle count is queried after each unblocking of the `waitForNextPeriod()` method. The time is queried directly after the cycle count is obtained, introducing a small constant error from the actual time when execution started.

4.2 Performance Criteria

The performance of the periodic processes will be evaluated by measuring interarrival times and latencies, as well as by the number of missed deadlines. We convert the interarrival results from cycles to the time units using our average clock frequency.

The distribution of latency and interarrival times for periodic processes reveal essential information about the behavior and robustness of an RTSJ implementation. Under quiescent conditions, it is expected that all three thread types will exhibit inter-arrival jitter less than 100 μ s and latency jitter less than 30 μ s. In addition, the RTSJ-specific threads should exhibit less inter-arrival and latency jitter than the Thread type, because the VM and underlying OS are specifically designed for the RTSJ thread types and the threads have more control over real-time priority. In the GP system state, we expected that it would be likely that both jitter measurements will increase slightly for the NHRT, but remain relatively small. The jitter results for RT and Thread are expected to increase significantly, into seconds and possibly minutes, because neither thread can preempt GC.

4.3 Results

Tables 1 and 2 and Fig. 1 through W052478.TMP summarize interarrival measurement results for periodic Thread, RT and NHRT process tests for the quiescent and GP system states. Tables 3 and 4 summarize the latency results for the periodic processes under quiescent and GP system states. For clarity the interarrival time graphs plot the positive and negative variation from the period.

Table 1. Periodic Thread Interarrival Times, Quiescent System State

	Thread	Realtime Thread	NoHeap Realtime Thread
Minimum	50.027 ms	49.367 ms	49.389 ms
Maximum	50.200 ms	50.633 ms	50.612 ms
Jitter	86.194 μ s	633.000 μ s	611.500 μ s
Mean	50.03 ms	50.0 ms	50.0 ms
Std. Dev.	3.088 μ s	15.133 μ s	13.768 μ s

Table 2. Periodic Thread Interarrival Times, GP System State

	Thread	Realtime Thread	NoHeap Realtime Thread
Minimum	50.120 ms	49.535 ms	49.382 ms
Maximum	2.825 s	5.500 s	50.618 ms
Jitter	1.388 s	2.726 s	618.000 μ s
Mean	416.7 ms	745.9 ms	50.0 ms
Std. Dev.	631.351 ms	1.091 s	18.218 μ s

Table 3. Periodic Thread Latencies, Quiescent System State

	Thread	Realtime Thread	NoHeap Realtime Thread
Minimum	2.860 ms	21.000 ^a μ s	17.000 μ s
Maximum	344.337 ms	655.000 μ s	633.000 μ s
Jitter	341.477 ms	634.000 μ s	616.000 μ s
Mean	173.4 ms	22.5 μ s	22.0 μ s
Std. Dev.	98.580 ms	10.717 μ s	9.767 μ s

^a The zeros are due to clock update granularity.

Table 4. Periodic Thread Latencies, GP System State

	Thread	Realtime Thread	NoHeap Realtime Thread
Minimum	25.764 s	52.900 s	400.051 ^b ms
Maximum	8.618 s	9.093 min	400.766 ms
Jitter	8.189 min	8.212 min	715.000 μ s
Mean	4.44 min	5.01 min	715.0 μ s
Std. Dev.	2.399 min	2.375 min	18.315 μ s

^b See footnote 3 on page 543.

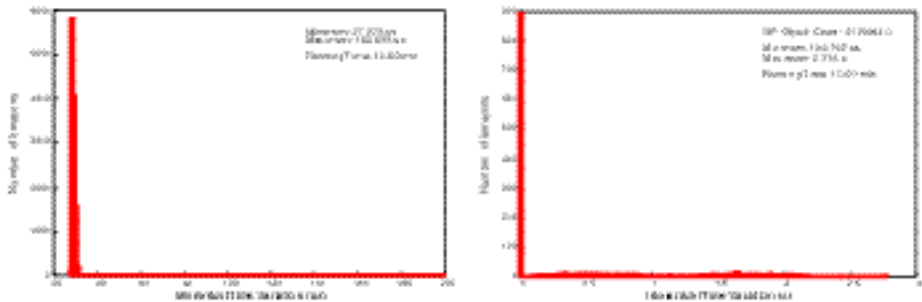


Fig. 1. Periodic Thread Process, Quiescent (left) and GP System States

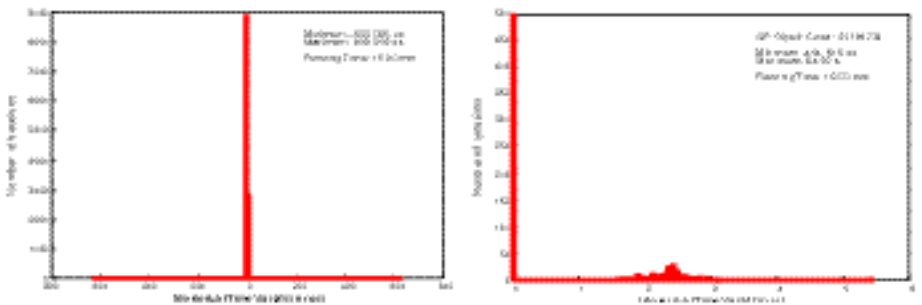


Fig. 2. Periodic Realtime Process, Quiescent (left) and GP System States

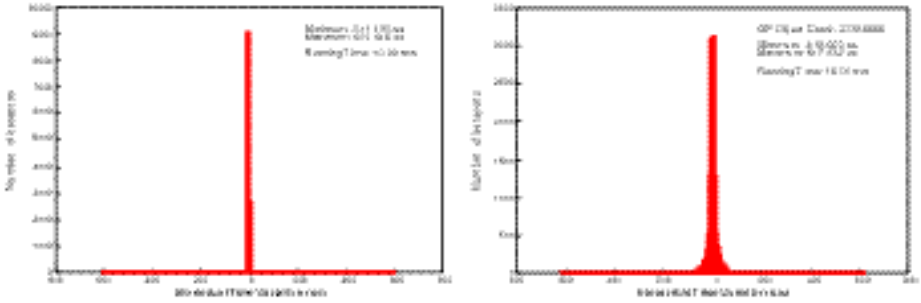


Fig.3. Periodic NoHeap Process, Quiescent (left) and GP System States

Note that 80 starting and 5 ending iterations were excluded in all latency and interarrival calculations. The starting iterations were not included to prevent anomalies due to cache interference and the ending iterations were excluded because notifying the periodic process of test end caused anomalous measurements. We plan to more fully document the effect of anomalies caused by cache and notification interference as our work proceeds.

4.4 Analysis

With a quiescent system state all three periodic processes produced similar results. None of the processes missed any deadlines and they all successfully completed the correct number of iterations with respect to the running time. Interarrival times occur most frequently near the actual period value. This is also demonstrated by a very low interarrival standard deviation value in the 3–16 μ s range. However, there are a small number of interarrival values that are $\pm 100 \mu$ s from the period value and a few over 600 μ s. The Thread based process exhibits minimal jitter because of the adjusted priority levels. Features of the TimeSys Linux kernel allow the priority of the Thread process to be the highest in the system. We hypothesize that because the Thread process has no overhead from the RTSJ implementation, it has less jitter than the RTSJ thread types. To verify this hypothesis, a periodic Thread test run was performed with the system priorities set to a level above the maximum user process priorities and we observed the expected higher jitter.

Under GP system state, the performance of the three periodic processes differ significantly. The periodic Thread reports interarrival times in the seconds and the successful iterations count is about 10,000 less than before. Latency times begin poorly and worsen to a magnitude of minutes. The periodic RT also exhibits deteriorating interarrival times and latencies. The poor performance of the Thread and RT is expected because neither thread has the ability to preempt the garbage collector that is exhaustively trying to clean up after the massive garbage producing threads.

We note, as expected, that computed jitter values, J_{lat} and J_{int} , for NHRT processes in all four tables are approximately the same but for Thread and RT processes they vary widely because executions are missing their deadlines by significant amounts of time.

The periodic NHRT still performs quite well with a GP system state. Interarrival jitter is only slightly worse. The increased activity of application threads increases the possibility of executing dynamically long critical paths through the kernel and therefore interrupts would be disabled for longer times, which would show up as increased jitter for the NHRT process.

Latency jitter has approximately the same magnitude for both systems states, but it is now offset by 400 ms³ and has a standard deviation about twice as large. The offset has been observed to be as large as second and is due to the fact that the very first period and first entry into the run() method does not get executed for some amount of time approximately equal to the mentioned offset. However, once the NHRT has started to run, it appears to be preempting GC, as expected. Appropriately, misses are detected shortly after the first execution, triggering the miss handler for the NHRT. This startup transient anomaly delays the execution affecting all subsequent periodic latency measurements.

4.5 Startup Anomaly

Wanting more insight into the startup anomalies mentioned in the last section we created another test type and plotted two graphs from the data. The test simply changed from a quiescent system to a GP system mid-way through the test run. On the graphs we also plot deadline misses and the times when the object producing threads produce another 50,000 unreferenced objects.

Figure 4 (left) displays the 30 seconds of a test run with an NHRT and a GP system state. On this run we reschedule the process as soon as possible after the first missed deadline and thus re-frame the relationship between period and execution, e.g., the third execution may now execute in period number 500 but after the re-framing we consider that third execution to be the 500th. Note that viewed in this manner the NHRT process behaves well over time but poorly in the initial 2 seconds.

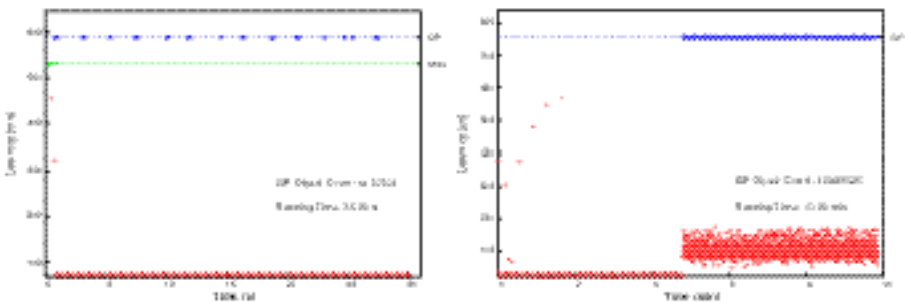


Fig. 4. Periodic NoHeap Process with Reschedule (left) and Delayed GP State

³ We have discussed this anomaly with the TimeSys development team and they agree that there seems to be an error in the JVM.

Figure 4 (right) shows nicely that the startup anomaly is due strictly to initiation of the NHRT process *while* the system is in GP state. Note the y-axis is in microseconds and no startup anomalies in the 400 ms range appear.

An important observation:

The startup anomaly appears unpredictably. Many runs do not show the anomaly, especially subsequent runs after an initial run within an instance of the JVM. When the anomaly does not appear latency values are similar to those in the last half of Figure 4 .

We do not understand the reasons for this observed behavior but we expect that, given the *predictability* of the process even in the presence of the anomaly the fix for this error is probably straightforward.

5 Periodic Timers

While both periodic and one shot timers are provided by the RTSJ, we currently have designed and executed tests for only the `PeriodicTimer`. The RTSJ notion of a periodic timer is different than one might expect. On expiration, these timers, by default, cause an associated piece of logic to be released for execution which typically are implemented with instances of `AsyncEventHandler`. The RTSJ expects that periodic timers will be used for application processes which might be termed, intermittently periodic. Since these processes are not well known in the literature an example here is appropriate. Consider a wheel control module of a Mars rover. While the rover is moving the control module will periodically monitor wheel position and issue motor commands. However, when the rover is stopped the wheel module has no purpose. Periodic timers are exactly the right abstraction to use because while the rover is stopped the timer is simply disabled, it retains all of its state, but demands zero processor cycles. Contrast use of a periodic thread: While the rover is stopped either the thread continues to unblock and discover there is nothing for it to do, or, its period has to be changed to some large value (and then changed back when the rover wants to move). In the first case the thread causes a demand on processor capacity and in the second, more complicated management code is required.

5.1 Procedure

A `PeriodicTimer` using a non-heap `BoundAsyncEventHandler` (BAEH) is tested under quiescent and GP system states for a fixed amount of time. A non-heap event handler is used, because it ensures that an NHRT will be used to execute the `run()` method of the event handler, allowing the handler to preempt GC. A BAEH is used because we did not want the time penalties of binding the event handler dynamically to a thread to affect our measurements. The setup for both system states is the same as presented for periodic threads, except that a periodic timer is created and started instead of a periodic thread.

Most of the settings held constant in the periodic thread test apply to this test as well. The start time and interval set at creation of the timer are 10 s and 50 ms,

respectively. The release parameters of the timer event, cost and deadline, also remained unchanged. However, in this test, no miss handler is defined. The priority of all related threads, such as the GP threads, have also been set to the same level, and the priority of the BAEH handling the timer events has a default value of `PriorityScheduler.MAX_PRIORITY` less 5. The level of garbage production is also controlled the same way as was done in the periodic thread test.

The runtime data collected in this test are for the most part identical to the data collected in the periodic thread test. The running time is equivalently the approximate amount of time from when the test timer is started to when the last timer event is handled. It has a typical value of 10 min. The number of successful iterations is the number of timer events handled by the BAEH. As in the periodic thread test, the total number of Integer objects created by GP threads is the GP object count. The cycle counter is queried each time the BAEH handles a timer event and a high resolution time is obtained at the start of execution of each event handler.

5.2 Performance Criteria

The performance of the `PeriodicTimer` will be evaluated by interarrival and latency measurements. The interarrival and latency calculations are performed in an identical manner to what was previously done in the periodic thread test, i.e., latency is the interval between release and the start of execution and interarrival time is the interval between subsequent executions. We expect that the `PeriodicTimer` with a non-heap BAEH will produce results similar to the periodic NHRT, under both system states.

5.3 Results

Tables 5 and 6 summarize the interarrival and latency measurements for both system states. Figures 5 and W052478.TMP display the complete test results as histograms.

Table 5. Periodic Timer Interarrival Measurement Summary

	Quiescent	GP
Minimum	49.367 ms	49.362
Maximum	50.633 ms	50.720 ms
Jitter	633.000 μs	679.000 μs
Mean	50.00 ms	50.00 ms
Std. Dev.	16.455 μs	60.498 μs

Table 6. Periodic Timer Latency Measurement Summary

	Quiescent	GP
Minimum	82.000 μs	450.145 ms
Maximum	716.000 μs	451.063 ms
Jitter	634.000 μs	918.000 μs
Mean	83.73 μs	450.35 ms
Std. Dev.	11.822 μs	57.925 μs

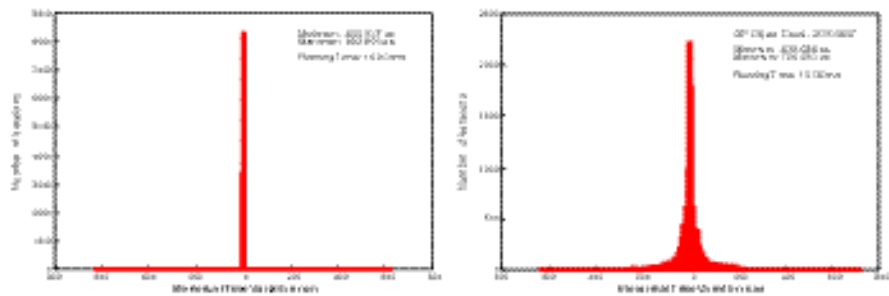


Fig. 5. Periodic Timer Interarrival, Quiescent (left) and GP System States

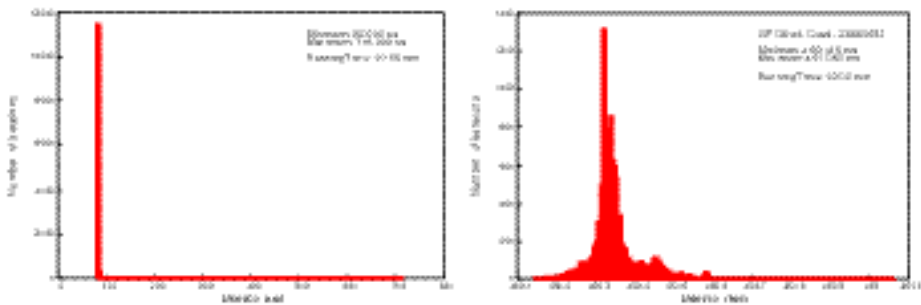


Fig. 6. Periodic Timer Latency, Quiescent (left) and GP System States

5.4 Analysis

For both system states, the periodic timer produced results similar to the NHRT periodic process, as expected. Note that the startup anomaly appears in these measurements as well. We note that the two methods of calculating jitter produce approximately equal values which we expected and saw in the results for the NHRT periodic process.

6 Asynchronous Events and Handlers

An AsyncEvent (AE) is an RTSJ class which abstracts and represents the notion of something happening and the AsyncEventHandler class holds logic which is associated with one or more AEs. When an AE occurs, as a result of a bound happening or a call to the AE's fire() method, associated handlers are released for execution.

6.1 Procedure

The purpose of these tests is to determine the latency between the time an AE occurs and its associated handler begins execution. We use an NHRT periodic process to repeatedly invoke AE.fire() and the associated handler is non-heap (thus we expect it to produce results similar to an NHRT process). The measurements are similar, i.e., the release time is the time AE.fire() completes and the start of execution time is when the handler begins to execute.

We set slightly different priorities in this test. Since it is the performance of the handler in which we are interested we assign it a priority value of PriorityScheduler.MAX_PRIORITY less 5, and the priority of the NHRT process less one.

6.2 Performance Criteria

We compute the latency exactly as in other tests. We expect the handler test to produce results similar to the NHRT periodic process in both system states.

6.3 Results

Table 7 shows a summary of our measurements of the asynchronous event handling system and Fig. 7 display all of our computed latencies.

Table 7. Asynchronous Event Handler Latency Summary

	Quiescent	GP
Minimum	57.000 μ s	129.000 μ s
Maximum	699.000 μ s	856.000 μ s
Jitter	642.000 μ s	727.000 μ s
Mean	58.14 μ s	232.8 μ s
Std. Dev.	6.248 μ s	39.040 μ s

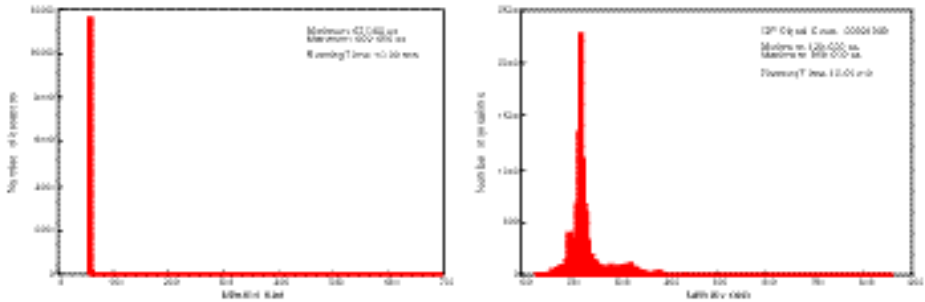


Fig. 7. Asynchronous Handler Latency, Quiescent (left) and GP System States

6.4 Analysis

The results are reasonably close to what we expected. We do note that the standard deviation for the two system states shows a greater difference than in all of the other tests of non-heap processes but is still reasonable.

7 Hard Real-Time Test

The RTSJ design considers hard real-time to mean that the temporal constraints, e.g., deadlines, expressed by an application are guaranteed to be met in all cases. The system must understand its own jitter characteristics (distribution, maximum, minimum, etc.) well enough to modify the feasibility analysis (maximum jitter can be included in the cost value for periodics) to be able to guarantee the validity of the results of the analysis. Since we have collected such values for JTime we hypothesized that we could, in fact, construct a system of periodic processes implemented by NHRT constructs, an associated feasibility analysis which included the jitter maximums and for which the result was success, and execute those processes, at a high actual utilization, without missing any deadlines.

To test our hypothesis constructed multiple NHRT periodic processes with the following code fragment as the body of their periodic loop

```
[1] while (true) {
[2]     count = 0;
[3]     start = getCycles();
[4]     while (count < cyclesPerMillisecond) {
[5]         count = getCycles() - start;
[6]     }
[7]     waitForNextPeriod();
[8] }
```

This loop body simulates a workload or cost of 1 ms. We gave these processes a period of 10 ms; thus each process requires 10% of the processor. We expected to be able to execute 6 of these processes given that our jitter (which include overhead) values for NHRT's with a GP system state are around 650 μ s in the worst case. A quick back-of-the-envelope feasibility analysis, including overhead, for 6 such processes would be successful because the actual processor utilization would be

$$6 \times 1.65 \text{ ms} / (10 \text{ ms}) \times 100 = 99\%$$

and the periods are harmonic (i.e., the same)⁴. We ran this test for 5 hours *with a GP system state* and *no* process missed its deadline. We can thus assert the specific claim that this implementation of the RTSJ can, indeed, support a hard real-time periodic process of 100 Hz with 60% utilization.⁵

⁴ A simple rate monotonic analysis or deadline monotonic analysis will show that this task set is feasible.

⁵ This is utilization as seen by the application and does not include overhead and jitter.

8 Conclusion

We consider the results presented in this paper to be astoundingly good. The predictability and low latencies achieved are unprecedented in the Java community.

A key issue needs to be examined: can the jitter values shown in our results be considered upper bounds or are they simply values associated with our particular tests? Of course, only TimeSys can answer that question competently, however, in private discussions between Sun Microsystems Laboratories and the TimeSys development team we believe that these jitter values can be considered in the order of the actual upper bounds.⁶

Thus, the RTSJ, as embodied by the the TimeSys implementation, is capable of executing applications which have hard real-time constraints. A comprehensive feasibility analysis for JTime would include maximum upper bounds on jitter in its calculations of feasibility and will, we claim, produce a correct analysis in that if the analysis asserts that the schedulable objects in the feasibility set will all meet their deadlines (with, of course, the requirement that the schedulable objects honor their given costs), then they will meet their deadlines, even considering any activity by Java threads and the garbage collection routines.

References

- [1] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull. The Real-Time Specification for Java. Addison-Wesley, 2000.
- [2] Angelo Corsaro and Douglas C. Schmidt. Evaluating Real-Time Java Features and Performance for Real-time Embedded Systems. Technical Report 2002-001, University of California, Irvine, 2002.
- [3] Kelvin Nielsen. Thoughts Regarding Sun's Real-Time Specification for Java, <http://www.j-consortium.org/rjwg/opengroup.1-02.pdf>
- [4] D. Sharp, et al. Evaluating Real-Time Java for Mission-Critical Large-Scale Embedded Systems, The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, pp. 30–36, 2003.
- [5] TimeSys. JTime 1.0. http://www.timesys.com/index.cfm?bdy=java_bdy.cfm, 2003.
- [6] TimeSys. TimeSys Linux/NET 4.0. http://www.timesys.com/index.cfm?bdy=linux_bdy.cfm, 2003.
- [7] <http://www.rtyj.org>, 2003

⁶ We make this assertion cautiously and realize that it is likely that even TimeSys may not be able to guarantee these bounds to 100%. However, we believe that significant effort has gone into the analysis. Also, we note, that these bounds depend on choices by the application environment developers, including but not limited to, the choice of device drivers installed in the TimeSys Linux RTOS (noting that poorly written drivers can cause significantly higher jitter).

Utility Accrual Scheduling with Real-Time Java

Shahrooz Feizabadi¹, William Beebe, Jr.², Binoy Ravindran¹,
Peng Li¹, and Martin Rinard²

¹ Real-time Systems Laboratory
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

{shahrooz,binoy,pe1i2}@vt.edu

² MIT Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{wbeebe,rinard}@mit.edu

Abstract. Graceful performance degradation during overload conditions is the primary objective of soft real-time systems. Utility accrual soft real-time scheduling algorithms allow specification of highly customized temporal system behavior during overload. Such algorithms are typically found in real-time *supervisory* systems where significant run-time uncertainty exists. This paper outlines an investigation of several utility accrual scheduling algorithms implemented in a Real-Time Java (RTJ) environment. These alternate schedulers are constructed, tested, and evaluated under the MIT FLEX/RTJ Compiler Infrastructure. The scheduling framework for this environment and its associated scheduling primitives are described and the corresponding performance characteristics are profiled. Furthermore, we outline the architecture of an experimental distributed Real-time Java scheduler.

1 Introduction

The Real-Time Specification for Java (RTSJ) mandates a strict fixed-priority preemptive scheduler with 28 unique priority levels [1]. This is consistent with core facilities traditionally provided by real-time operating systems. The properties of this scheduling model are well-understood and widely utilized [2,3]. Augmented with built-in support for priority inversion avoidance [1], RTSJ provides the essential tools for hard real-time programming. This *base scheduler* is the minimal RTJ scheduler implementation requirement.

Beyond the base scheduler, the specifications anticipate construction of alternate schedulers and provide the requisite API's to accommodate them: real-time threads now enjoy language-level support for such dynamic scheduling notions as deadline, cost, and cost enforcement. The motivation for this paper is to investigate the viability of construction of such dynamic schedulers in general, and Utility Accrual (UA) schedulers in particular.

Contributions

Below is an outline of the contributions of this paper:

Utility Accrual RTJ scheduling: We demonstrate the viability of constructing UA schedulers in Real-time Java. By corollary, we demonstrate the implementation of complex dynamic deadline schedulers. We present the performance measurements for these algorithms in a FLEX/RTJ environment, compare the results with their counterparts in a QNX RTOS environment, and note the associated performance characteristics.

User-Level RTJ Thread multiplexor: The FLEX/RTJ `chooseThread` is designed to offer fine-grain thread control at the user-level: any thread can be arbitrarily chosen for execution at any point for an arbitrary period of time. Presenting a single point-of-entry into the system, the thread multiplexor is simple and effective.

Separation of policy and mechanism: Inherent complexities of interactions between the JVM and the OS kernel present significant technical challenges for implementation of alternate scheduling policies at the user level. JVM's typically implement a thin scheduling layer over the existing OS scheduler, map Java threads onto native threads, and thereby leverage the properties of the underlying scheduler.

The FLEX/RTJ scheduling framework abstracts away implementation complexities by providing a high-level construct for transparent thread manipulation. It allows the programmer to focus on policy rather than implementation mechanisms.

Experimental Distributed RTJ Thread Scheduler: We outline a framework for scheduling of distributed threads in a real-time Java environment. This is a `chooseThread` implementation over sockets and includes the basic scheduling API's required for a distributed system.

Implementation: Concurrency on a uniprocessor system is merely a high-level abstraction – albeit an effective one. The thread multiplexor is in essence a serializer. To utilize it, the programmer needs to adopt a “systems-level perspective” in place of the traditional higher level concurrency constructs. We discuss insights and experience gained from implementation of our schedulers in this environment.

2 Utility Accrual Scheduling

Real-time systems, by definition, are those designed to predictably adhere to a predefined set of timeliness constraints [4,5]. *Hard* real-time systems must always

satisfy all predefined timeliness requirements. *Soft* real-time systems, by contrast, are those designed to exhibit a specific temporal behavior when the system is unable to meet all its timing constraints.

The most commonly used notion of a timing constraint is the *deadline*. A deadline is a point in time prior to which, the completion of an activity yields the most utility; and conversely, less utility is derived by completion of the task after that specific point in time. In the context of hard real-time systems, this is the binary choice of completing a task by the deadline and gaining its full utility, or gaining no utility for task completion after the deadline [4]. Failing to move a robot arm out of the way of the next coming job in an assembly line, for instance, would amount to significant *negative* utility.

In soft real-time systems, the utility of a task is a function of its completion time. Completion of a task after the deadline may yet be of some utility — a radar “ghost” resulting from delayed processing of residual data is preferable to a blank screen.

An emerging class of real-time control systems, collectively known as *supervisory* systems [6], are increasingly incorporating soft real-time algorithms for resource scheduling decision support. Supervisory systems are typically comprised of a hierarchy of lower level real-time systems, and function in complex environments with significant run-time uncertainty.

In the absence of a priori knowledge of upper bounds on task execution times, the hard optimality criterion of always meeting all deadlines is difficult to guarantee at design time. As soft real-time systems enter overload conditions, the “sequencing optimality” can then be defined in terms of producing a specific system-wide behavior such as minimizing missed deadlines, maximizing aggregate utility, etc.

The timing constraints of such supervisory systems can be accurately expressed in terms of Time/Utility Functions (TUF’s) [4]. Figure 1 illustrates several examples of TUF’s. This generalization offers a great degree of versatility for representing timeliness requirements. Deadline timing constraints can be viewed as a special case, and modeled using rectangular TUF’s – Figure 1(b).

Supervisory systems are specifically designed to operate in environments where there exists a potential for overload. TUF-based algorithms can be found in a wide range of application domains such as telecommunications, defense, and industrial automation [7]. Utility accrual models can also aid management and decision support systems: during a catastrophic natural disaster, an overloaded emergency response center can temporarily delay dispatching of emergency supplies to a remote location in favor of first servicing a nearby high population density area.

Supervisory system design is a significant software (and systems) engineering effort and requires global knowledge of dynamically varying conditions to appropriately compose the corresponding TUF’s. Examples of highly sophisticated UA-based supervisory systems in complex environments include MITRE Corporation’s AWACS tracking system [8], and the *BM/C²* System developed by General Dynamics and Carnegie Mellon University [9].

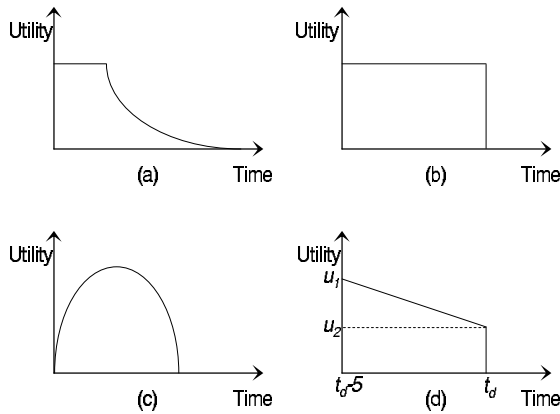


Fig. 1. TUF examples

Figure 1(d) illustrates the data processing TUF for two sensors 180 degrees out of phase, each with a 10 second period. The sensor data processing deadline is therefore 10 seconds. If, however, the data from one sensor is processed within the first 5 seconds of the period, it can improve the quality of the data received from the out of phase sensor. Completion of data processing beyond the first 5 seconds can still partially strengthen the incoming data from the out of phase sensor. The utility of the data, however, linearly diminishes to zero with the approaching 10 second deadline as new sensor data will be collected [8]. A more detailed UA algorithm design and implementation process is outline in [9].

3 The Scheduling Framework

Although the RTSJ explicitly supports scheduling policies other than fixed-priority preemptive, it does not elaborate the interfaces to allow the creation of such schedulers. It was envisioned that OS/JVM vendors would implement alternate schedulers and make them available to programmers through API's allowed by RTSJ. Implementation of such schedulers would require access to internals of the JVM and the underlying OS [10]. The design of the FLEX/RTJ scheduling framework bridges this gap and alleviates the need for detailed knowledge of, and access to, the JVM or the OS.

3.1 FLEX/RTJ

We implemented the general real-time scheduler framework and user threads package using the MIT FLEX compiler infrastructure. FLEX is an ahead-of-time compiler for Java that generates both native code and C. The resulting binaries can be executed as stand-alone applications requiring only POSIX compliance from the underlying OS.

Real-Time Java is a superset of the Java language (RTSJ) which provides a framework for building real-time systems. FLEX has an implementation of Real-Time Java which includes both region-based memory management [11,12,13,14] and a scheduler framework. Separation of policy from mechanism to achieve flexibility in real-time user threads packages is not a new idea [15].

Our threads package exports thread multiplexing and lock handling (which influences scheduling policies) to the user scheduler. User schedulers can be written entirely in Java. Since the `chooseThread` abstraction can implement any interleaving of threads, many policies can be implemented simply by providing a single method. Performance of the system is often dependent on the performance of the `chooseThread` method. The system restricts interaction with the scheduler to a minimal interface, enhancing productivity by reducing the scope of the interactions which must be considered when debugging or optimizing an implementation of a new scheduling algorithm. For example, a simple pre-empting round-robin policy can be expressed in only four lines of code using `chooseThread`.

3.2 Design Benefits

The primary design benefit of the scheduler framework is to provide a single, central point of control for all timing aspects of the Real-Time Java system. Without centralized information and control, a single event can invalidate any hard real-time guarantees of the system. For instance, a scheduler may choose to defer handling an asynchronous event or asynchronous interrupt in favor of running a higher-priority task. The scheduler may need to know about all timers in the system in order to perform adequate admission control. Priority inheritance relies on knowledge of locks held in the system.

A second design benefit is to provide a single point where knowledge and control of the operating system, Real-Time Java libraries, the Java runtime, the user program, middleware, and information from static program analysis can be brought together. For instance, the user program or middleware may inform a specialized user scheduler of unique scheduling characteristics of an asynchronous event (which may prove useful to an implementation of the *Distributed Real-Time Specification for Java*. [16])

A third design benefit is to promote concise and flexible scheduler design through the use of minimal interfaces with expressive power. Since the scheduler may potentially need to manage information about many different aspects of the system simultaneously, complex, sophisticated schedulers which take advantage of the full flexibility of the system are often hard to write. Concise schedulers are easier for programmers to write and debug. Furthermore, the system provides a debugging interface which logs every event occurring in the system, every choice made by the scheduler, and the state maintained by the scheduler which led to a particular choice. A utility provides a graphical summary of the choices made by the scheduler and the time interval which has elapsed between each scheduling point, a simple overview of scheduler behavior. Since the scheduler is simply a

Provided by user scheduler:

chooseThread(time) $\rightarrow \{ \text{thread}, \text{quanta} \}$

Chooses a user **thread** from the currently available threads to run for the next **quanta** of time or until blocked, whichever occurs first. The scheduler multiplexes user threads into a heavy thread. The **time** provided to **chooseThread** is the start of the context switch.

feasibility

Standard RTSJ interface to determine feasibility of a set of tasks.

Utilities provided by system:

getDefaultScheduler() $\rightarrow \{ \text{scheduler} \}$

To install a scheduler, simply point this to your **scheduler**.

contextSwitch()

Force a context switch at the earliest available time.

setQuanta(time)

Reset the time until the next context switch. **setQuanta** is provided to adjust the current schedule in response to events.

clock() $\rightarrow \text{time}$

Return process CPU **time** used by the process for calculating work done by a thread.

sleep(time)

Yield control to the operating system scheduler for **time**.

Optional RTOS-specific utilities provided by system:

reserveCPU/NET $\rightarrow \text{success flag}$

Interface to TimeSys linux to provide CPU/NET reservation access to scheduler. The returned **flag** indicates whether the reservation was successful. Successive calls to **reserveCPU/NET** simply modify the current reservation.

Fig. 2. Scheduler interface

Java class, a scheduler can even be tested outside of the system during initial development.

3.3 System Facilities

The system provides several primitives to facilitate the implementation of hard and soft real-time scheduling policies as described in Fig. 2. Work and system load can be estimated using **clock**. A reservation interface allows schedulers to interact with the TimeSys kernel to provide hard real-time guarantees when running under a TimeSys RTOS. **setQuanta** provides preemption capabilities and **sleep** can yield time to the kernel. **contextSwitch** can force a context switch in response to handling an event.

The system can generate many events which can affect policy decisions (Fig. 3). The scheduler can choose to either handle or ignore them. POSIX thread events are generated by native methods or by the runtime in response to Java synchronization. All POSIX thread events can be handled by the sched-

Mandatory event handlers provided by user scheduler:**addThreadremoveThread(thread)**

Called by the system to give the scheduler an opportunity to update internal data when a Java or C thread starts or ends.

enableThreaddisableThread(thread)

Called by system to inform scheduler of threads unblocking or blocking on locks.

Optional event handlers for making policy decisions:

pthread events

Handlers are provided to allow the scheduler to be notified of all pthread events handled by our user-threads package.

RTSJ events

Handlers are provided to allow the scheduler to be notified of all events generated by our RTSJ implementation.

Policy-specific events support:

user program events

The scheduler can provide its own API to the user program or middleware for generating events directly (via a simple method call).

compiler generated events

The compiler can weave code into the user program to call methods on the scheduler directly based on program analysis simply by inserting **CALL** instructions into the program during compilation.

Fig. 3. Scheduler event handler interface

uler. RTSJ events are generated by our RTSJ implementation and can inform the scheduler of all aspects of the RTSJ which influence scheduling. Events generated by the user program, compiler, or middleware are all policy-specific. Thread entry and exit, blocking and unblocking cannot be ignored by the scheduler since they impact the active thread list.

3.4 Distributed Real-Time Java Threads

A socket-based implementation provides distributed event support (Fig. 4). **generateDistributedEvent** can generate an event on another scheduler across the network. The event can be handled by **handleDistributedEvent**. In the socket-based implementation, a client thread and a server thread are scheduled by the scheduler. The server thread causes **disableThread** to be called on the scheduler when no pending request is available. **enableThread** informs the scheduler that an event has arrived. The client thread causes **disableThread** to be called when a request has been sent. The scheduler can then use **chooseThread** and **setQuanta** to provide hard real-time bounds on the processor time given to servicing the network. We have implemented an RMI-based mechanism that allows communication between multiple RTJ schedulers each running on remote node.

Utilities provided by the system:

bind(scheduler name) → thread

Bind the current scheduler to **scheduler name** in the name service and start a server to handle events. Returns the thread which handles incoming events. Network timing can be managed through **chooseThread**, **disableThread**, and **enableThread**.

resolve(scheduler name) → scheduler stub

Resolve the **scheduler name** in the name service to a stub which represents the destination scheduler.

generateDistributedEvent(destination, message ID, data) → thread

Create an event on the **destination** scheduler using a network call. Returns the thread which handles outgoing events.

Optional event handler for distributed real-time scheduling:

handleDistributedEvent(name, message ID, data)

Allow the scheduler to respond to an event generated by another scheduler.

Fig. 4. Distributed real-time thread and event support

3.5 Implementation

The thread multiplexor uses a standard, signal-based, **sigsetjmp** and **siglongjmp** implementation. The system tells the kernel to use signal-interruptable kernel calls. The system saves the thread context using **sigsetjmp**. The system then sets a timer at the end of a context switch to cause a **SIGALRM** to be generated after an interval set by **setQuanta**. The signal handler calls **chooseThread** to determine the next thread to be run, retrieves the saved registers associated with the thread and uses **siglongjmp** to simultaneously unblock **SIGALRM** and restore the program counter to that saved by the context switch.

Thread start involves setting up a jump buffer and environment that appears like a **sigsetjmp** from the thread's start. Blocking on a lock calls **disableThread** to inform the scheduler of the blocked status and then forces a context switch by signalling **SIGALRM**. Unblocking calls **enableThread**. **SIGALRM** is blocked during scheduler calls to prevent reentry, facilitate local reasoning about the correctness of the policy implementation, and provide bounds on stack usage. Therefore, the scheduler implementation should provide bounds on method calls to ensure bounded preemption latency.

4 Scheduler Performance

In this section we present an overview of the implemented scheduling algorithms followed by the associated performance results.

4.1 Experiment Design

The following points regarding our experimental setup are of significance:

- The algorithms we investigated were initially coded and tested in a simulated environment. The simulated scheduler performance depicts ideal conditions under various processor loads as the simulator assumes instantaneous preemption, and no scheduler or OS overhead.
- Each scheduler was tested with a set of concurrent periodic tasks and its behavior at each scheduling point was verified against a manual trace for various execution hyper-periods and load conditions.
- The tasks in the system are vacuous threads executing busy wait to generate artificial CPU load.
- The experiments were performed with garbage collection disabled.
- We tested the schedulers with the highest possible priority under the TimeSys Linux RTOS to minimize OS overhead and eliminate unanticipated CPU contention from other processes — alternatively, it is possible to use the TimeSys CPU reservations mechanism to guarantee each scheduler a dedicated share of the CPU as in [17].
- We then tested the same scheduling policies with identical task/load conditions in a QNX Neutrino RTOS environment using C/POSIX implementations.
- The performance of each scheduler is then compared to its simulated, and QNX counterparts.

4.2 EDF

The Earliest Deadline First (EDF) scheduling algorithm merely executes tasks in ascending order of deadline at any scheduling point [18].

This is a dynamic scheduling policy as the tasks' absolute deadlines are determined at run-time and the execution order of tasks are arranged accordingly.

EDF is shown to be optimal (up to 100% CPU utilization) for uniprocessor systems: if a task misses a deadline under an EDF scheduler, no other policy can arrange a schedule where the task meets its deadline [18].

EDF suffers from a “domino effect” [19] at or near overload. The point at which EDF fails, however,

can be used as a performance indicator of system overhead. RED (Robust Earliest Deadline Scheduling) [20] addresses the instability of EDF during overload and uses the mechanism of “deadline tolerance” for enhanced scheduling decision making. The performance of our EDF implementation is depicted in Fig. 5.

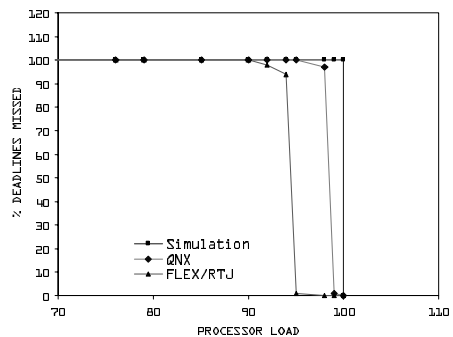


Fig. 5. Performance of EDF scheduler

1. Create an empty schedule.
2. Sort all tasks by deadline.
3. For each task (in increasing order of deadline):
 - 3.1. Insert task in schedule at deadline position.
 - 3.2. Check schedule feasibility.
 - 3.3. While (schedule not feasible and schedule not empty):
 - 3.3.1 Remove task with lowest potential utility density from schedule.
4. Execute task in schedule with earliest deadline.

Fig. 6. Sketch of LBESA algorithm

4.3 LBESA

LBESA (Locke’s Best Effort Scheduling Algorithm) [19] was the first publicly available utility accrual real-time scheduler intended for supervisory control systems. LBESA was implemented in the Alpha distributed real-time OS kernel [21]. The BM/C^2 system [9] was developed using Alpha.

LBESA uses non-convex functions (values cannot increase after a decrease) to describe time utility. Furthermore, it stochastically characterizes task execution times using probability distribution functions.

Task interarrival times need not necessarily be periodic or deterministic. Tasks are assumed to be independent. A general overview of the algorithm is outlined in Fig. 6.

The algorithm produces an EDF-based absolute deadline ordering and determines feasibility. If the system is in overload, the algorithm continually rejects the task with the lowest potential utility density until a feasible schedule is achieved.

The concept of potential utility density used by LBESA is analogous to the notion of “return on investment.” At any arbitrary point, the time invested thus far in the task’s execution can be weighed against the potential return on investment at maturity - utility yield at task completion.

Reflecting the nature of its application domain, a full LBESA implementation is proportionally complex and requires significant system resources. We implemented the basic algorithm for the special case of rectangular TUF’s and known worst case execution times. The scheduler performance is illustrated in Fig. 7.

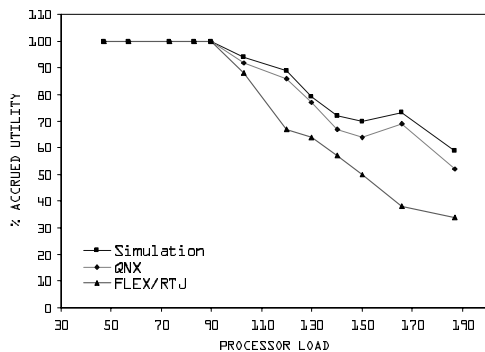


Fig. 7. Performance of LBESA

In this experiment, the performance of the QNX implementation closely follows the simulation results, whereas the RTJ implementation somewhat lags behind. Due to higher aggregate system overhead and lower time resolution, the RTJ implementation “slips” early-on and determines a tentative schedule infeasible, whereas the QNX implementation is able to make the deadline. The subsequent scheduling decisions by each system therefore vary, resulting in the performance disparity.

4.4 DASA

DASA (Dependent Activity Scheduling Algorithm) [22] is another example of a utility accrual resource scheduler used in supervisory control systems. The second generation of the Alpha OS includes an implementation of DASA. The Open Group’s OSF.1 MK7.3a [23] distributed RTOS also incorporates DASA. The DASA algorithm has the twofold objectives of maximizing system-wide accrued utility while minimizing the number of missed deadlines.

Concurrent tasks can develop relative dependencies as they serially access devices, channels, and other exclusively shared system resources: a task requesting a resource is considered dependent on the task currently holding that resource. Given its objectives, DASA makes appropriate scheduling decisions while taking into account such dynamic dependencies, and the corresponding precedence relationships. The algorithm assumes that possible deadlock resulting from cycles in the dependency graphs can be detected and resolved. A simplified version of the algorithm, DASA/ND [24], can be used in systems where tasks are known to be independent. The performance of the DASA/ND scheduler is illustrated in Fig. 8.

DASA considers all deadlines as hard, and exclusively relies upon rectangular utility functions. Utility maximization at any scheduling point is achieved by dynamically determining the task that would yield the most utility if chosen to continue. The remaining competing tasks may then be preempted or aborted (during overload) in favor of the chosen task. DASA is equivalent to EDF (i.e., is optimal) up to the theoretical bound of 100% CPU utilization [22]. Figure 9 is a generalized sketch of the algorithm.

Tasks in DASA are assumed to contain the static deterministic attributes of worst case execution time, absolute deadline, and utility. Furthermore, the timing requirements of the shared resources need to be known in advance: should

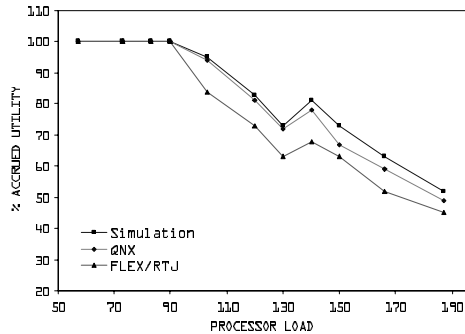


Fig. 8. Performance of DASA/ND

1. Create an empty schedule.
2. Determine dependencies among tasks.
3. Calculate potential utility density for each task.
4. Resolve deadlocks if detected.
5. Sort tasks by potential utility densities.
6. Examine each task in decreasing order of potential utility density:
 - 6.1. Tentatively add the task and its dependent tasks to the schedule in deadline order.
 - 6.2. Test schedule feasibility.
 - 6.3. If not feasible, remove task from schedule.
 - 6.4. Optimize schedule if possible.
7. Execute task in schedule with earliest deadline.

Fig. 9. Sketch of DASA algorithm

a task be selected for abortion, the scheduler must know how soon the held resources can be released and reassigned.

Figure 10 shows the performance of a limited DASA implementation. Only one shared system resource is simulated to eliminate the possibility of deadlock, and tasks are assumed to implicitly release the resource at the conclusion of their execution.

As with LBESA, the performance disparity between the RTJ and QNX implementations is due to the relative overhead incurred by each system. The schedulers make identical decisions up to the point of nearing system performance limits. Though the task attributes (e.g., release time, execution profile, potential utility, etc.) are the same for each implementation, the schedulers make differing decisions after the initial “slip” caused by reaching performance boundaries. The RTJ implementation does, however, continue to accrue utility as best it can, given the circumstances of its environment.

Furthermore, it should be noted that the aggregate accrued utility of each system is highly dependent on the present task-sets and their associated attributes — it is possible (though not likely) for a task-set to produce identical ordering of scheduling decisions by each implementation during overload.

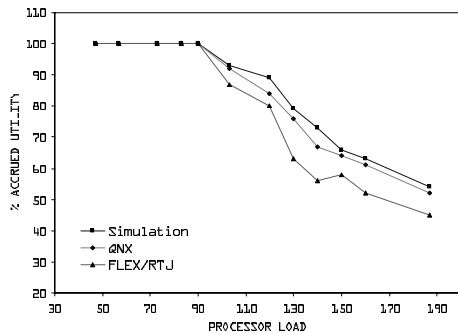


Fig. 10. Performance of DASA

5 Conclusions

Our implementations and subsequent performance analyses demonstrate the viability of complex utility accrual scheduling in a Real-time Java environment. We outline the implemented algorithms and compare their respective performance measures.

The FLEX/RTJ thread multiplexor scheduling primitive provides an effective and flexible tool for construction of arbitrary schedulers. This abstraction now allows RTJ scheduler construction to be a user-level activity rather than a static JVM/OS component.

The scheduling framework has been extended to accommodate distributed thread management. The corresponding high-level scheduling abstractions are designed to parallel the simplicity and effectiveness of the stand-alone framework. Preliminary tests of the experimental system indicate the viability of extending the existing scheduling model.

FLEX has been ported to a variety of architectures such as x86, StrongARM, and PowerPC under operating systems such as RedHat, Debian, Familiar, and TimeSys Linux. Users can compile to alternate targets using C cross-compilers. The FLEX/RTJ framework provides an effective array of tools for embedded systems design and implementation.

References

1. G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, 2000. Latest version available from <http://www.rti.org>.
2. M. Klein, T. Raylya, B. Pollak, R. Obenza, and M. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Norwell, Massachusetts: Kluwer Academic Publishers, 1993.
3. G. Buttazzo. *Hard Real-time Computing Systems : Predictable Scheduling Algorithms and Applications* Boston, Massachusetts: Kluwer Academic Publishers, 1997.
4. E. Douglas Jensen *Real-Time for the Real World* <http://www.real-time.org/deadlines.htm>
5. A. Burns, and A. Wellings *Real-Time Systems and Programming languages* London, UK: Addison Wesley, 2001
6. E. D. Jensen *Asynchronous Decentralized Real-Time Computer Systems* In *Real-Time Computing Proc. of NATO Advanced Study Institute*, Springer Verlag, October 1992.
7. E. D. Jensen, and B. Ravindran *Guest Editor's Introduction to Special Section on Asynchronous Real-Time Distributed Systems* In *IEEE Transactions on Computers*, IEEE Computer Society August, 2002.
8. R. Clark and E. D. Jensen and A. Kanevsky and J. Maurer and P. Wallace and T. Wheeler and Y. Zhang and D. Wells and T. Lawrence and P. Hurley *An Adaptive, Distributed Airborne Tracking System* In *Proceedings of The Seventh IEEE International Workshop on Parallel and Distributed Real-Time Systems* Springer-Verlag, April 1999

9. D. Maynard, S. Shipman, R. Clark, J. Northcutt, R. Kegley, B. Zimmerman, and P. Keleher. "An example real-time command, control, and battle management application for Alpha." Technical report, CMU Computer Science Dept., December 1988. Archons Project Technical Report 88121.
10. P. Dibble. *Real-Time Java Platform Programming* Paulo Alto, California: Sun Microsystems Press, 2002.
11. W. Beebee, Jr. Region-based memory management for Real-Time Java. MEng thesis, Massachusetts Institute of Technology, September 2001.
12. W. Beebee, Jr. and M. Rinard. An implementation of scoped memory for Real-Time Java. In *First International Workshop on Embedded Software (EMSOFT)*, October 2001.
13. A. Sălcianu, C. Boyapati, W. Beebee, Jr., and M. Rinard. A type system for safe region-based memory management in Real-Time Java. Technical Report TR-869, MIT Laboratory for Computer Science, November 2002.
14. C. Boyapati, A. Sălcianu, W. Beebee, Jr., and M. Rinard. Ownership types for safe region-based memory management in Real-Time Java. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, June 2003.
15. Thorsten Kramp. FREE JAZZ: An User-Level Real-Time Threads Package Designed for Flexibility SFB 501 Report, October 1998.
16. *Distributed Real-Time Specification for Java* <http://www.drtsj.org>
17. D. de Niz, and R. Rajkumar. *Chocolate: A Reservation-Based Real-Time Java Environment on Windows NT* In Proceedings of the IEEE Real-time Technology and Applications Symposium Washington D.C., June 2000
18. C. Liu, and J. Layland. "Scheduling Algorithms for Multiprocessing in a Hard-Real-Time Environment". JACM, vol. 20, pp. 46–61, January 1973.
19. C. Locke. Best-Effort Decision Making for Real-Time Scheduling. PhD thesis, Carnegie Mellon University, 1986. CMU-CS-86-134.
20. G. Buttazzo, and J. Stankovic. *RED: Robust Earliest Deadline Scheduling* In Proceedings of the 3rd International Workshop on Responsive Computing Systems pages 100–111, September 1993.
21. E. Jensen and, J. Northcutt. *Alpha: A Non-Proprietary Operating System for Large, Complex, Distributed Real-Time Systems* In Proceedings of The IEEE Workshop on Experimental Distributed Systems Pages 35–41, 1990.
22. R. Clark Scheduling Dependent Real-Time Activities. PhD thesis, Carnegie Mellon University, 1990. CMU-CS-90-155.
23. MK7.3a Release Notes *The Open Group Research Institute* <http://www.real-time.org/docs/RelNotes7.Book.pdf>
24. P. Li, B. Ravindran, and T. Hegazy. "Implementation and Evaluation of a Best-Effort Scheduling Algorithm in an Embedded Real-Time System". In *Proc. 2001 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 22–29, Tucson, Arizona, November 4–6, 2001.

A C++ Native Interface for Interpreted JVMs

Angelo Corsaro¹ and Corrado Santoro²

¹ Washington University

Department of Computer Science and Engineering
1 Brookings Drive, BOX 1045, St. Louis, 63130 Missouri, USA
`corsaro@cse.wustl.edu`

² University of Catania

Dept. of Computer and Telecommunication Engineering
Viale A. Doria, 6 - 95125 - Catania, Italy
`csanto@diit.unict.it`

Abstract. This paper describes JuNI++, a C++ native interface for interpreted Java Virtual Machines. While JuNI++ was initially designed in order to integrate the RTSJ implementation *jRate* with Juice, a virtual machine for small footprint environments, its engineering and performance advantages outlived the initial goal. The main contribution of this paper is to show how a C++ based native interface can fully and efficiently support the Java mapping in interpreted environments without requiring any cooperation from the C++ compiler.

1 Introduction

Most of Java execution environments for embedded systems are featured by a pre-compiler/pre-processor that transforms bytecoded classes into a different representation form which is, in general, more compact and optimized. This is made necessary due to the limited availability of resources of embedded systems (limited memory size, above all), thus requiring class files to be transformed into a form that contains only the parts fundamental for program execution. For example, Sun's KVM [20], Superwaba [3] (the JVM for PDAs) and IBM's WebSphere Studio Device Developer [14] have a tool that produces a single file containing the class closure of an application; it is generated by removing debugging information and performing a pre-linking and a pre-verification process. To improve performances, other Java environments pre-compile classes into native code (*ahead-of-time* compilers), thus producing a single native executable. This is common in real-time environments where meeting time-constraints often requires high execution efficiency. This approach is used by *jRate* [9], the Real-Time Specification for Java (RTSJ) [6] implementation developed by one of the authors, which is based on the GNU Compiler for Java (GCJ) [13]. All of these mechanisms work well when the whole code (program and libraries) is directly available from a ROM or a flash memory, and class loading from external sources is thus not needed. However, most of the modern hardware platforms for embedded systems and Systems-On-Chip (SoC) are equipped with network

connections that make these systems ready to be integrated in – more or less standardized [4,5] – distributed environments, such as the Internet or Internet-based (TCP/IP-based) networks. Java applications running on these systems can take advantage of network class loading to retrieve programs, at boot, from a suitable code repository (thus avoiding software updating problems occurring when using *e.g.* ROMs), or to enable class loading on-demand during program execution (*i.e.* by means of the `Class.forName()` method). Java execution environments as those above are not suited and need to be integrated with modules able to perform loading, linking and execution of classical “class” Java files.

To this aim, in this paper we investigate how to integrate Java native libraries, written for an ahead-of-time compiler such as GCJ using the Cygnus Native Interface (CNI), into an interpreted JVM. The reasons that led us to start such a research derive from the need to integrate `jRate` in an environment based on a light executive – called *NUXI* [17,2], developed by one of the authors – equipped with an interpreted J2ME [18] virtual machine – called *Juice* – designed for that executive. The aim is to provide a system developer with a complete solution for the design of a real-time embedded system not only capable to run ROMized/pre-compiled Java programs, but also suited for Java class loading and execution on-demand.

To reach such an objective, the main issue to deal with is the integration of the CNI [12] into *Juice*. CNI is the interface to native classes provided by GCJ and used by `jRate`; it is based on C++ types, C++ class definitions and implementation, and is essentially designed for compiled environments¹. *Juice* instead provides a pure-C native interface, called *JuNI* (*Juice Native Interface*). Therefore, in order minimize the changes needed to port `jRate` on *Juice*, *JuNI++* has been designed and implemented, so to support calling of native methods implemented in C++ from interpreted code and vice versa. The resulting virtual machine has been called *Juice++*. It is worth noticing, how having a C++-like native interface gives several advantages in terms of code reuse, maintainability, and integration with existing C++ code.

The paper is structured as follows, Section 1 motivates our work; Section 2 provides an overview of *Juice* and `jRate`; Section 3 introduces *Juice++* and shows how we have addressed the challenges that arise in integrating *Juice* and `jRate`; Section 4 provides some comments on the proposed approach, also comparing it with other proposals; finally Section 5 gives our conclusions provides an outline of the future works.

2 Juice and jRate

2.1 Juice

Juice is a POSIX-compliant J2ME-CLDC interpreted JVM. It was designed to be integrated in *NUXI* [17,2], a light multi-tasking executive for Intel-based

¹ Basically a native class is defined and implemented as a C++ class, provided that it uses suitably defined C++ types that corresponds to Java types.

platforms, developed for both educational purpose and embedded/real-time applications. The aim is to obtain a complete solution – operating system and JVM – for a Java-enabled embedded system. Indeed, we investigated some other existing solutions before deciding to write a new virtual machine from scratch. We had initially chosen KVM [20], the Sun’s J2ME-CLDC reference implementation, but, although the developed port worked well in NUXI, it was not suitable for our next step: to implement Java real-time services. KVM has a thread scheduler which does not use OS-native multi-threading and concurrency control services, but it simply performs a manual context switch each time it interprets 1000 bytecodes². This is obviously time/CPU-dependent and thus not a good solution for time-constrained environments. We considered other virtual machines, such as Superwaba [3] and Kaffe [1], but the former is too PDA-oriented while the latter is not J2ME and its implementation is too complex for our purpose (porting to NUXI and adaptation to J2ME should need too many patches and thus too much work).

For these reasons, we decided to start to develop a new virtual machine, *Juice*, designed keeping in mind two basic requirements: execution on POSIX multi-threaded environments (as NUXI is) and tight integration with the executive. These objectives are met by using pthread-based services to support Java multi-threading (each Java thread is mapped to a system thread) and concurrency control (synchronized constructs mapped to mutexes and wait/notify methods implemented with condition variables). This implies a minimal overhead because there is a direct mapping with OS-native mechanisms. Since the targets of Juice are embedded systems, optimization, footprint and performances are key characteristics, which have been considered during development. Size reduction is achieved by loading only class parts which are strictly needed (debug information are not loaded, for example) and by hashing all strings in order to avoid redundancy and to speed-up comparisons (if two strings have the same hash pointer then the strings are the same).

2.2 jRate

jRate³ is an open-source RTSJ-based real-time Java implementation that we are developing at the Washington University, St. Louis (WashU). jRate extends the open-source GCJ front-end and runtime system [13] to provide an ahead-of-time compiled platform for the development of RTSJ-compliant applications. jRate differs from a traditional JVM since there is no bytecode interpretation. Instead, jRate ahead-of-time compiles RTSJ applications into native code.

jRate uses Generative Programming (GP) [10] in order to make it possible to have a configurable, customizable and yet efficient RTSJ implementation. GP is also used as a way of exploring design alternatives, which differ from the RTSJ, in a well engineered manner. The generative behaviour is achieved by using a

² This number is indeed a factor scaled to thread’s priority.

³ jRate can be freely downloaded at <http://tao.doc.wustl.edu/~corsaro/jRate>

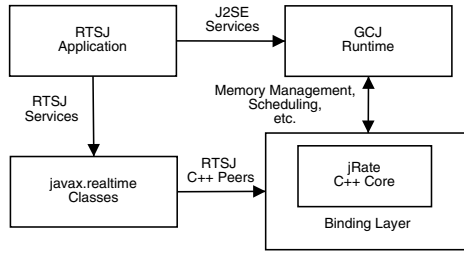


Fig. 1. The jRate-Core and RTSJ binding

series of techniques and tools such as C++ template meta programming [10], AspectJ [21] and Python [16] scripting.

In order to maintain the *jRate* runtime reusable, configurable, and highly efficient, *jRate* relies on a set of RTSJ-like C++ Template classes. This set of C++ Template classes, called the *jRate-Core*, provide a configurable kernel which can be reused in different settings and language binding by properly instantiating the template classes. As depicted in Fig. 1, RTSJ applications developed using *jRate* rely on the GCJ runtime for basic Java services, and on the *jRate* runtime for RTSJ services. In order to make it easier to reuse the *jRate* code base, most of the extension that were needed by the GCJ runtime have been factored out and moved to the *jRate-Core*. As shown in Fig. 1, the RTSJ binding represents just one instance of use of this C++ core, and by writing the proper binding, RTSJ-like abstraction could be provided to other languages such as C#, C++ etc.

Currently *jRate* supports most of the RTSJ features such as memory areas, real-time threads, periodic threads, asynchronous event handlers, timers, etc. It allows a fine control over the properties of the different types of memory areas, such as size, allocators, locking etc. *jRate* provides several optimizations such a constant time memory reference checking, and single parent rule test implementation [7] and lock free dispatching of events [8].

3 Juice++: Integrating Juice and jRate

One of the issues that arise when using *jRate* is that the runtime footprint of GCJ based applications is currently quite high. This is due to the fact that GCJ libraries are currently too monolithic and hard to subset. This inflexibility badly matches the requirements of small-end embedded systems, in which JVM and application have to take on the order of kilobytes of memory. Thus, *jRate* on GCJ is not currently a viable solution for very small footprint devices, for which memory constraints are more stringent than the runtime efficiency. On the other end, Juice was designed to be a small footprint JVM (it also run on NUXI which is well suited for devices with limited memory resources). Based on what said so far, it should seem natural that the way to go to provide a uniform solution across different embedded systems classes is that of porting the *jRate* runtime system to Juice.

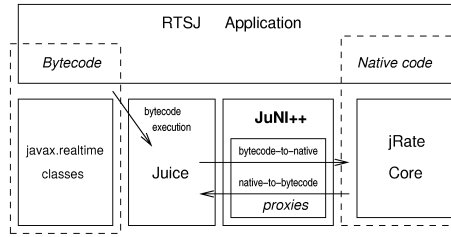


Fig. 2. Juice++ Architecture

3.1 Requirements for Porting the jRate Runtime

A key requirement to make the porting of jRate practically effortless, is that the target JVM supports CNI, which is GCJ's preferred native interface. The choice of using CNI in jRate was dictated by several factors, some of which include performances and programmability – CNI allows to use a classical C++ programming style for the development of native methods and provides better performances than Sun's Java Native Interface (JNI) [19].

The approach we have decided to follow is that of designing a CNI-like native interface for Juice, called *JuNI++* (*Juice Native Interface*). In such a design work, we faced several challenges since, (1) differently from CNI, we did not want to require a Java-aware C++ compiler as GCJ is, and (2) we wanted to support a full mapping from Java to C++, also adding some features, such as support for interfaces, which are not currently provided by CNI. Shortly, our objective was to have Java and C++ interoperate in a safe, efficient and seaming-less manner. In order to achieve this goal, the main issues which have to be addressed can be classified in two categories: *Java-to-C++ mapping* issues and *Java-to-C++ interoperability* issues. The Java-to-C++ mapping is rather important since it has to provide an intuitive and easy to program environment. On the other hand the mechanism chosen to achieve Java-to-C++ interoperability has to be efficient in both time and space, otherwise it would not be usable in embedded systems with stringent time and space constraints. Next we provide an overview of JuNI++, as well an overview of the issues we had to solve to make it possible to support a CNI-like native interface without requiring cooperation from the C++ compiler.

3.2 JuNI++

Figure 2 depicts the structure of the solution adopted by showing the modules composing Juice++⁴, their relationships and the role of JuNI++. We consider a generic (RTSJ) application which is composed by Java and native portions and the jRate runtime. Given that bytecode is interpreted by the Juice virtual machine, the JuNI++ module provides the appropriate services to perform *access*

⁴ Juice++ is the JVM resulting from the integration of JuNI++ into Juice

Table 1. Java-to-C++ Mapping

Java	C++
class	class
package	namespace
reference	pointer
operator .	operator →
char	jchar ¹
int	jint ¹
long	jlong ¹
float	jfloat ¹
double	jdouble ¹
Java Exception	C++ Exception

¹ These types are defined by means of **typedef** statements on the basis of the target platform and compiler.

to native code (i.e. native method invocation), from the bytecode interpreter, and *access to bytecode* (i.e. bytecoded method invocation, reading/writing object attributes) from native code. This is made possible by means of “proxies”, which are pieces of code acting as bridges and suitably produced by the *JuNI++ Proxy Generator (JuniPG)*, given the class files constituting the application and the needed libraries. It is worth noticing, that the proxies need to be generated for all those classes that are explicitly used in the native code – even if these classes do not implement native methods. This is needed since the native code has to be able to call, from the C++ side, the method defined in these classes.

Writing native code using JuNI++ is rather straightforward since the Java language features can be seen (to a certain extent) as a proper subset of the C++ language. To this extent, Table 1 reports the correspondence between Java and C++ constructs and types.

In order to understand what is the Java-to-C++ programming model provided by JuNI++ let's consider a simple example. Let's assume that we are implementing a **Counter** class, and that we want to write some methods of this class natively (this is indeed an artificial example, but it is meant to explore most of the JuNI++ features in a contained way). Let's then assume that the **Counter** class, whose code is listed in Listing 1, provides some methods implemented directly in Java and some other methods declared as native, such as **inc()**, **dec()** and so on. It also implements the **Comparable** interface, and throws an exception in case the counter's count becomes either negative or exceeds a certain maximum value.

Listing 2 contains the code generated by the JuniPG. As it can be easily seen from the listing, a **sample::Counter** C++ class corresponds to the **sample.Counter** Java class. The former class is declared within a *namespace* which matches the package in which the Java class was declared. Something worth noticing is that none of the C++ class methods declare any exception. This is done so that C++ counter part of Java classes can throw Java unchecked exceptions. The last thing that should be noticed in Listing 2 is the declaration

Listing 1: Java code for the Counter sample

```

package sample;

class Counter implements Comparable {
    private int count, maxCount;

    public Counter(int maxCount) {
        this.maxCount = maxCount;
    }
    public void reset() {
        this.count = 0;
    }
    public native void resetAndInc();
    public native void inc() throws CountOverflowException;
    public native void dec() throws CounterUnderflowException;
    public native int getCount();
    public native void setCount(int newCount);
    public int compareTo(Object o) {
        this.compareTo((Counter)o);
    }
    public int compareTo(Counter c) {
        (this.count < c.count) ? -1
                               : (this.count == c.count) ? 0 : 1;
    }
}

```

of *proxy methods*. As it will be shown next, these methods provide all the glue code that ties together the Java and the C++ world.

Listing 3 contains the implementation of the native methods. As shown in this listing, the JuNI++ native interface provides a quite natural way of mapping Java native methods to C++.

While the example shown so far seems to be quite simple, there are actually quite a few issues that need to be taken care of in order to make Java and C++ interoperate in an efficiently and seamless manner. So far we have seen how to implement native methods, but we have not revealed how Java code can actually invoke methods on C++ classes and vice-versa. Next Section will provide an explanation of the techniques used by JuNI++ to achieve this goal. In our proposal we tried to come up with a solution that, as much as it could, would not rely on the C++ object layout, and on specific features provided by a given C++ compiler. This implies a high degree of portability was a big concern taken into account in designing our solution.

3.3 Making Java Talk with C++

The first problem that has to be solved is that of making it possible, for the JVM, to invoke C++ methods that implement native code. As widely known, the mechanism used by JVM to identify a method is based on its “signature” which is obtained through the operand associated with `invokeXXX` opcodes⁵ (see [15]);

⁵ `invokevirtual`, `invokestatic`, `invokespecial`, and `invokeinterface`.

Listing 2: C++ mapping for the Java simple Counter

```

#include <junipp/junipp.h>
#include <java/lang/Comparable.h>

namespace sample { class Counter; }

class sample::Counter :
    public ::java::lang::Object,
    public ::java::lang::Comparable {
public:
    Counter(jint);
    void resetAndInc();
    void inc();
    void dec();
    jint getCount();
    void setCount(jint);
    int compareTo(::java::lang::Object* o);
    int compareTo(::sample::Counter* c);
private:
    jint count;
    jint maxCount;
public: /* Proxy Methods */
    virtual jvalue getField(int index, int depth);
    virtual void setField(int index, int depth, jvalue val);
    virtual void invokeMethod(int index, int depth, jenv env);
    jvalue getField0();
    void setField0(jvalue value);
    void invokeMethod0(jenv env);
    void invokeMethod1(jenv env);
    void invokeMethod2(jenv env);
    void invokeMethod3(jenv env);
    void invokeMethod4(jenv env);
protected:
    static FieldSetter_t fieldSetterTable[];
    static FieldGetter_t fieldGetterTable[];
    static MethodCaller_t methodCallerTable[];
};

```

Listing 3: C++ mapping for the Java simple Counter

```

void sample::Counter::inc() {
    this->count++;
    if (this->count > this->maxCount)
        throw CountOverflowException;
}

void sample::Counter::resetAndInc() {
    this->reset();
    this->count++;
}

void sample::Counter::setCount (jint newCount) {
    this->count = newCount;
}

```

the problem is thus to provide an efficient and effective way to directly access a C++ method starting from a string Java signature.

Our solution is strictly C++ based and uses a variation of the *Proxy* design pattern [11] to provide a translation layer between Java-style and C++-style method invocations: this is made possible by means of the *proxy methods* generated by JuniPG (see Listing 2). When Juice++ executes a “new” opcode, it

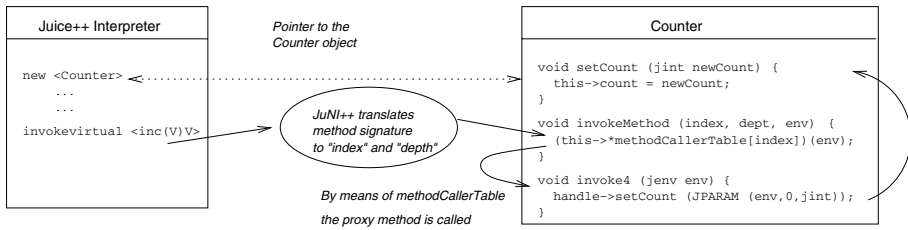


Fig. 3. Proxy Objects and Java-to-C++ Access

passes the control to JuNI++ which creates the proper C++ objects. Figure 3 clarifies this mechanism with reference to the `Counter` example: there, the execution of “`new`” triggers the creation of an instance of the C++ class `Counter`. Thanks to its structure, the C++ object created provides a quick and effective way to access native C++ implementation. The mechanism is based on a static table, the `methodCallerTable` in Listing 2, which contains the pointers to some proxy methods which are called `invokeMethod0()`, `invokeMethod1()`, etc., each associated and implementing the call to a native method. More specifically, the steps performed by the JVM when a “`invokevirtual`” (or “`invokestatic`”) opcode is executed are described below, and also illustrated by the Figure 3 with reference to the `Counter` example (let us consider, for the moment, that the called method belongs to the object’s class and not to one of its ancestors):

1. A numerical `methodIndex` is derived by the method signature. This corresponds to the index of the method in the method area of the class;
2. The `invokevirtual` bytecode is replaced with `invokevirtual_quick`, with the operand equals to `methodIndex`. This operation avoids another string lookup and speed-up the next execution of this call (indeed this kind of optimization is performed in many virtual machines, as suggested in [15]);
3. The JVM prepares the method parameters, which are marshaled in a structure of the `jenv` type – this is the “method environment”, a structure containing data needed for the execution of method, *i.e.* the parameters, the local frame, the operand stack, a reference to the current thread, etc.
4. The method `invokeMethod()` of the C++ object is called, passing as parameters the `methodIndex` and the method environment (the meaning of the “`depth`” parameter will be instead explained later on);
5. The proxy method `invokeX()` is called by means of an indirect addressing on the basis of the dispatch table `methodCallerTable`;
6. The proxy method calls the native method. The proxy method extracts the parameters from the method environment, by using the `JPARAM` macro, and finally calls the native method of the C++ object.

If there is an inheritance tree and the method invoked is not implemented in the class’ object but in one of its ancestors, the step 1 of the sequence above does not succeed as the requested method cannot be found in that class. In this case, several solution are possible to find the right method, some of which trade space for time efficiency.

The most elegant but, not necessarily the most time efficient, is based on recursion. During the execution of the step 1 above, the JVM recursively searches for the method signature in the ancestor classes and, each time a hierarchy level is passed, a `depth` parameter is incremented by one. When the method is found, this parameter represents the depth of the inheritance hierarchy, starting from the class of the referenced object, for the class which implements the searched method. The depth parameter determined is passed to the `invokeMethod()` of a proxy; this method checks if depth is zero – meaning that the implementation of the native method is in the associated class – the steps 5 and 6 above are directly executed; if it is non-zero, the `invokeMethod()` of the ancestor C++ class is called with the depth decremented by one. The process is thus executed recursively until the depth reaches zero, meaning that the class implementing the requested method was found.

Using the described technique, execution time depends on the value of the depth parameter. A constant time solution could be instead easily implemented by using the depth as an index into an *array of pointers* which refer to the dispatch tables of parent classes. In this case, the cost to pay is the use of more memory space than the former solution, since each class must include the array of pointers in addition to its own dispatch table.

3.4 Accessing Attributes

Another issue concerning Java-to-C++ interoperability is the access to object attributes. According to the programming model of JuNI++/CNI, each attribute declared in a Java class is directly mapped to an attribute declared in the C++ native implementation (see attribute `count` of the Counter example). This allows C++ native methods to access attributes using the standard “->” operator (see Listing 2), but when the JVM needs to read or write such an attribute (by means of “`getField`” / “`putField`” opcodes), a suitable technique able to reach a C++ attribute starting from its Java string signature is needed.

This problem is quite similar to that of invoking native methods and is therefore solved using a technique based to that of Sect. 3.3. As Listing 2 reports, the C++ object has two static pointer tables, `fieldGetterTable` and `fieldSetterTable`, generated by JuniPG. They contain the pointer to the proxy methods `getField0()`, `setField0()`, `getField1()`, `setField1()`, etc. Therefore, like native method invocation, the JVM, when executing “`getField`” / “`putField`” opcodes, determines the “`fieldIndex`” and “`depth`” parameters, and then invokes the method `getField()`/`putField()` of the C++ object. The latter calls the proxy method (by using the pointer table) which, in turn, performs read/write of the relevant object field.

3.5 Handling Interfaces

The Java language allows multiple inheritance of interfaces, thus their treatment in JuNI++ deserves some further discussion – multiple inheritance usually complicates method dispatching. Java interfaces can only *define constant values*

(*static final attributes*) and *declare methods*: interfaces are treated by JuNI++ as regular C++ classes, transforming the `implements` clause in a standard C++ inheritance (see the example in Listing 2 which shows how is declared class `Counter` which, in Java, implements the interface `java.lang.Comparable`).

Constant values thus do not present particular issues: if a Java interface `IntA` defines a constant `foo`, JuniPG will generate a C++ class, called `IntA`, defining the static attribute `foo`. The latter can thus be accessed from native code with the usual C++ mode `IntA::foo`.

As for methods are concerned, their definitions appear in the constant pool of the class implementing the interface, and thus can be treated as regular methods with the mechanism described in Sect. 3.3. Only the use of the “quick” variant of the `invokeinterface` opcode requires particular care, since the object implementing an interface could not be always the same and the index of the method to be invoked thus varies on the basis of the class of the target object. In the current prototype version of JuNI++ this problem is solved as suggested in the Sun’s JVM specification [15].

3.6 Making C++ Talk with Java

The last issue to be dealt with, is related to the invocation of Java methods from native code. Since the aim is to provide a totally C++-based native programming environment, the problem is to allow native code to invoke a Java method with the usual statement “*object->method.name (params)*”. Listing 3 shows an example in which a native method – `resetAndInc()` – invokes a method – `reset()` – which is written in Java (see also Listing 1).

Making C++ talk with Java is much easier than the reverse. The C++-to-Java interoperability can be obtained, by using proxies which provide the proper interface. Specifically, for each Java class defining a method called by some native code, the JuniPG automatically generates the associated C++ class and embeds in it a method with the same name and prototype. This C++ method is indeed a proxy that prepares the local parameters, invokes the Java method and gathers a return value, if present. Listing 4 reports an example of such a proxy code for the `Counter` example. It shows the code generated to provide access to Java method `reset()`: it first creates the environment needed for method execution, then invokes the Java method through a call to the JVM interpreter⁶ and finally releases the memory held by the `jenv` variable since it is no more used.

4 Discussion

The work described in this paper demonstrates how it is possible to embed into a JVM, designed for bytecode interpretation, some native code written using a programming model typical of (ahead-of-time) compiled environments. Even if

⁶ `sample.Counter.reset_V.V` is the pointer to the hashed string containing the signature of the Java method; it is generated by the JuniPG.

Listing 4: A C++ proxy accessing a Java method

```

void sample::Counter::reset() {
    jenv * env = newJavaEnvironment (this);
    callJavaMethodFromCXX (this, sample_Counter_reset_V_V, env);
    destroyJavaEnvironment (env);
}

```

our work derives from the necessity to solve a specific problem – porting `jRate` to an embedded environment – it presents several interesting aspects and provides also some advantages with respect to other approaches used in building JVMs and native interfaces. It is worth noticing that the main characteristic of our approach is that it provides a way to integrate C++ code within Java programs; this gives two advantages on native code development and execution: (i) a *more simple and standard programming model*, and (ii) a *performance improvement*.

As for the former item, using C++ means to provide a standard way to write native code and does not force the programmer to use JVM-specific services to access Java methods or attributes from the native part implementation. Anyone who has written native methods knows how verbose and cumbersome is Sun’s JNI [19]. On the other hand, with JuNI++, invoking a method is as easy as: “*object->method (params)*”. This basically means that the programmer can ignore the services provided by the native interface of the JVM employed, and the knowledge of Java classes of the API and the developed application, together with the names of C++ types to be used to represent Java types⁷, suffices to immediately start writing native code. Obvious consequences are *reuse* and *portability*; the former is due to the fact that re-engineering or upgrade of the native interface do not have any effect on native code, since only the stub generator (JuniPG) needs to be re-written to reflect the new structure/functionality of the native interface, while native code remains the same. Portability implies the possibility to use the same native code on different virtual machines, provided that they are equipped with the appropriate stub generator: indeed our approach in designing the native interface is quite general and can be adapted to any virtual machine, thus enabling native code portability.

As far as performance improvement is concerned, even if we still have not performed appropriate measures⁸, we expect that our approach provides better performances than JNI. Indeed, if we analyze the way in which JNI and JuNI++ implement Java-to-native and native-to-Java interoperability, we find that (1) Java-to-native interoperability roughly needs the same amount of operations in both JNI and JuNI++ (method finding, parameter preparation in the stack or in a structure, calling to the native method), (2) native-to-Java interoperability should definitively be more efficient in JuNI++ – attributes and methods are directly accessed (from native code) without the necessity to use interface functions as described above. Indeed, we plan to perform, in the near future, some

⁷ i.e. `jint` for `int`, `jchar` for `char`, etc.

⁸ This will be a subject of our future works as well as code optimization.

measurement tests in order to validate what we assert and optimize – when needed – the code of JuNI++.

5 Conclusions

This paper proposed a software architecture that allows seamless integration of native code written in C++, into Juice, an interpreted POSIX-compliant Java Virtual Machine developed to be used in a small executive designed for embedded systems. The proposed architecture basically served to support the porting of the jRate RTSJ implementation into Juice, thus providing a complete solution for a Java-enabled real-time embedded environment. To this aim, we designed a native interface, called JuNI++, aimed at providing the layer needed to perform Java/C++ integration. This is done by using *proxy methods* which bridge Java code with C++ native code. Proxy methods are provided by means of a suitable code generator, called JuniPG, which builds the interface code starting from the class files of the Java application. We called Juice++ the JVM resulting from the introduction of JuNI++ into Juice.

As for future works, we plan to improve Juice++ by analyzing its characteristics in terms of performances and memory footprint. Our aim is to allow the introduction of optimizations on the basis of the requirements of the target environment, *e.g.* for size, if the target has limited memory resources, or speed, in the case of applications needing fast execution. This will be done means of options passed to JuniPG which will generate the proxy code, suitably optimized for the desired requirements.

References

1. Kaffe Virtual Machine Home Page. <http://www.kaffe.org>, 2002.
2. NUXI Home Page. <http://nuxi.iit.unict.it>, 2002.
3. Superwaba Home Page. <http://www.superwaba.org>, 2002.
4. Antonella Di Stefano and Corrado Santoro. A Java Kernel for Embedded Systems in Distributed Process Control. *IEEE Concurrency*, , special issue on “Operating Systems Research & Development”, 8(4), 2000.
5. Antonella Di Stefano and Corrado Santoro. Java Devices for Distributed Process Control. In *Proc. of 2000 IEEE International Symposium on Industrial Electronics (ISIE 2000)*. IEEE, 2000.
6. Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, 2000.
7. A. Corsaro and R. K. Cytron. Efficient Memory-Reference Checks for Real-Time Java. In *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 51–58. ACM Press, 2003.
8. A. Corsaro and D. C. Schmidt. Evaluating Real-Time Java Features and Performance for Real-time Embedded Systems. In *Proceedings of the 8th IEEE Real-Time Technology and Applications Symposium*, San Jose, Sept. 2002. IEEE.
9. A. Corsaro and D. C. Schmidt. The Design and Performance of the jRate Real-Time Java Implementation. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 900–921, Berlin, 2002. Lecture Notes in Computer Science 2519, Springer Verlag.

10. K. Czaenwcki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, Massachusetts, 2000.
11. E. Gamma, R. Helm, R. Johnson, and R. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994.
12. GNU is Not Unix. The Cygnus Native Interface for C++/Java Integration. <http://gcc.gnu.org/java/papers/cni/t1.html>.
13. GNU is Not Unix. GCJ: The GNU Compiler for Java. <http://gcc.gnu.org/java>, 2002.
14. IBM. WebSphere Studio Device Developer. <http://www-3.ibm.com/software/wireless/wsdd/>, 2003.
15. T. Lindholm and F. Yellin. *The Java(TM) Virtual Machine Specification*. Addison-Wesley, 2nd edition, 1999.
16. M. Lutz. *Programming Python*. O'Reilly, 2nd edition, 2001.
17. C. Santoro. *An Operating System in a Nutshell*. Internal Report, Dept. of Computer Engineering and Telecommunication, UniCT, Italy, 2002.
18. Sun Microsystems inc. Java Micro Edition Documentation. <http://java.sun.com/j2me>, 2002.
19. Sun Microsystems inc. Java Native Interface Documentation. <http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html>, 2002.
20. Sun Microsystems inc. KVM White Paper. <http://java.sun.com>, 2002.
21. The AspectJ Organization. Aspect-Oriented Programming for Java. <http://www.aspectj.org>, 2001.

WORM CoRe 2003 PC Co-chairs' Message

Welcome to the Proceedings of the Workshop on Regulatory Ontologies and the Modeling of Complaint Regulations (WORM CoRe). This book reflects the issues raised and presented during the WORM CoRe workshop which proves to be an interdisciplinary forum for subject matters involving modeling ontologies of cross-border business regulations. WORM CoRe has a special interest in gathering researchers and practitioners from multiple disciplines (such as Law, Business, Ontologies, leg/legal-XML, ADR/ODR-XML, Tomography) and will be of interest to a wide range of researchers and people.

It may be the first time that such an outstanding selection of high-quality papers on the subject was brought together. We feel that these Proceedings will inspire further research and create an intense following. This first year, 17 papers were submitted and 9 were selected. Every paper was reviewed by at least one knowledge (/ ontology) expert, at least one legal expert, and some application-oriented experts. The papers were judged according to their originality, validity, significance to theory and practice, readability and organization, and relevancy to the workshop topics and beyond. The Program Committee comprised: Aldo Gangemi, Brian Hutchinson, Bruce Spencer, Gerd Wagner, Joost Breuker, Mariano Fernandez Lopez, Michael Sintek, Robert Meersman, Rita Temmerman, Said Tabet, Sophie Louveaux, York Sure, and Yves Pouillet.

We would like to express our deepest appreciation to the authors of the submitted papers and thank all the workshop attendees and the program committee members for their dedication and assistance in creating our program and turning the workshop into a success. Producing this book would not have been possible without the much appreciated contribution of Kwong Lai. Finally, we gratefully acknowledge the partial support of the EU project CCFORM (IST-2001-34908, 5th framework) that was crucial to establish this workshop.

Thank you and we hope you will enjoy the papers as much as we do.

August 2003

Mustafa Jarrar, Vrije Universiteit Brussel, Belgium
Anne Salaun, e-consult, Belgium
(Workshop on Regulatory Ontologies and the Modeling of
Complaint Regulations 2003 Program Committee Co-chairs)

On the Ontology of Digital Identification

Giles Hogben, Marc Wilikens, and Ioannis Vakalis

Institute for the Security and Protection of the Citizen
Joint Research Centre of the EC
Via Enrico Fermi 1, Ispra 21020 VA, Italy
giles.hogben@jrc.it

Abstract. Existing technical and legal definitions of identification and closely related privacy concepts show a confused and often circular semantics, in particular when applied to a digital environment. We examine the ontology of digital identification in the wider context of privacy. We begin with a formal definition of the ‘identical’ relation between 2 nyms and from this we derive a quantifiable notion of identification based on linkability and its opposite, anonymity. We base our logical model on a 3 layered semantic model theory. The results of this modeling show the context dependence of identification. Identification has meaning only in relation to a set of individuals known as the anonymity set, and an existing knowledge base of facts about these individuals.

1 Introduction and Survey of Current Models of Identity

Digital identities are fast becoming the most important and coveted assets in the information society. For example, the US Federal Trade Commission estimates that complaints about identity theft doubled in 2002. However, a survey of regulatory descriptions of identity reveals a lot of confusion. The US patriot act, 2001 [1], mentions identity and identification 72 times, without once attempting to define them.

The EU Data Protection Directive 95/46/EC [2] refers to identity as follows:

1. “an identifiable person is one who can be identified directly or indirectly by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.”
2. “to determine whether a person is identifiable, account should be taken of all the means likely reasonably to be used ... to identify the said person”

Technically, both definitions are circular as they use the words “identified” and “identity” to define identifiable. Also they do not define e.g. a “social” or “physiological” identity. Furthermore, while the Directive in principle applies to the processing of personal data and extensively uses the concept of personal identification in defining crucial legislative criteria, it does not explain the key mechanisms of personal identification. This should not be seen as a failing because the Directive aims to give a generic framework while leaving detailed implementation issues to context specific interpretations. For example in [3], the Article 29 Data Protection Working Party considers IP addresses as personal data in most end-user Internet surfing contexts because ISP's can identify Internet users to whom they have attributed IP

addresses by analysing logs. On the other hand, a third party could discover the dynamic IP address of a user but not be able to link it to personal data, but as stated in [3], the possibility nevertheless exists of linking the user's IP address to personal data, by invisible processing means such as cookies.

The W3C standard for automated privacy management, P3P also shows considerable inconsistency in its expression of identity issues. Consider the following phrase used in the specification: "In order to consider the data 'anonymized', there must be no reasonable way for the entity or a third party to attach the collected data to the identity of a natural person" (What is meant by "attach"? what is a natural person?).

Because concepts of personal data and identification are subject to different interpretations by involved stakeholders (e.g. ISP, data subject, standards body) there is a need for formalised description frameworks, which may be useful both in legal frameworks and in building technical architectures with formal machine readable ontologies based on this foundation. Formal ontology syntax such as DAML/OIL [4], RDFS [5] and OWL [6] can express clarified conceptual models with a well defined relationship to legal and end-user conceptual models and a syntax readable by software applications. For a fuller discussion of the usage of these ontologies within the specific domain of privacy and identity, see [7].

Some work has been done informally on the ontology of identification, for example in [8][9] and [10]. According [10], there are two conflicting issues: offering people anonymity with multiple pseudonymous identities, and on the other hand protecting business interests and reliability.[11] also describes the protection of privacy in transactions in terms of a choice between identification of clients on one side and protection of anonymity on the other. In [10] we find the following definition:

human identification is the association of data with a particular human being.

But identification is often entirely unrelated to the data subject or human being. Furthermore, in an electronic context, identities used by human beings are often far more abstract than the physical characteristics of their bodies. Although the idea of identifying somebody with a physical presence is appealing, we will show that the concept of identification is far broader and more complex.

[8] examines digital pseudonyms and credentials and the concepts of identity management are described in the context of multilateral security. The proposed identity management scheme is based on the concept of the partial and full identities of a user. The other paper [9] represents a considerable effort to define rigorously terms like anonymity, pseudonymity, unlinkability and unobservability, which are fundamental to the modelling of data protection transactions and use-case scenarios. The terminology is based on a framework of a sender sending messages to recipients in a network environment. [9] defines *anonymity* as:

The state of being not identifiable within a set of subjects, the ANONYMITY SET.

Anonymity is defined only with respect to a set of possible candidates for identity. We support the notion of relative anonymity, but the nature of the "subjects" referred to is not clear. The data subject is defined as "somebody who may take action". Including the concept of action in anonymity seems an unnecessary step. Furthermore, the concept of a full identity has little meaning given the context dependence of identity we describe.

We build on the above work to provide a clarified model of IDENTITY, consistent with its usage in legal contexts, but also logically rigorous and capable of describing and clarifying the scenarios which are inconsistent and circular in other models. Whilst it is the general thrust of research to try to protect privacy by regulating the collection of certain well-defined data types, we demonstrate that privacy sensitivity is not a property intrinsic to data but is derived in relation to a specific context.

2 Semantic Model Theory

In common with the formal ontology community and in particular, the RDF Model Theory adopted by the W3C [12], we assume a 3-layered semantics¹ with a many-to-many relationship between layers, as defined by “reference”. The layers are:

1. **Names:** Names are used to refer to concepts. In RDF semantics, they are called URIs and Literals. E.g. “mum” and “mom” are names referring to the concept MOTHER. We use the term “nym” for names used in identification contexts.
2. **Concepts:** Concepts are mental constructs which stand independently of the names which refer to them. For example the Italian Name “Cibo” and the English name “Food” both refer to the same concept. It is crucial to this model that two names may refer to the same concept. Note also that in the electronic context, persons such as GILES HOGGEN are seen primarily as concepts. Key concepts and relations of our model are denoted henceforth by capital letters. This convention is also used to differentiate between a CONCEPT and a NAME in examples.
3. **World:** Direct sensory experience. Two concepts are identical if they refer to the same real world set of experiences. For example GEORGE BUSH and PRESIDENT are not identical because PRESIDENT also includes experiences of BILL CLINTON.

For example, “George Bush”, “Dubaya” and his social security number are pseudonymous names, standing for the concept of GEORGE BUSH. Layer 3, the world layer, is a set of experiences (referred to by the concept layer) which can only be indicated demonstratively e.g. “hey isn't that GEORGE BUSH” – “Yes I think it is”. Although it is usual in ontological discourse for a person to be defined as an individual instance, we depart from this convention. A persona (person in an electronic context) is a far more arbitrary entity than in normal social discourse. Although it may include experiences of a physical presence, it also includes a much wider gamut of experiences from religious affiliation and skin color to blog entries, chat room responses and mailing list archives. It is therefore more accurate and convenient to see a person in such contexts as a concept binding together a set of instances (which may be attributed to a different “person as concept”).

As further support for this semantic model theory, we refer to cognitive science models [13], the philosophical work of Fodor [14] on the Language of Thought Hypothesis and the work of Chomsky [15].

¹ We take the model of RDF as a reference example and not a definitive model.

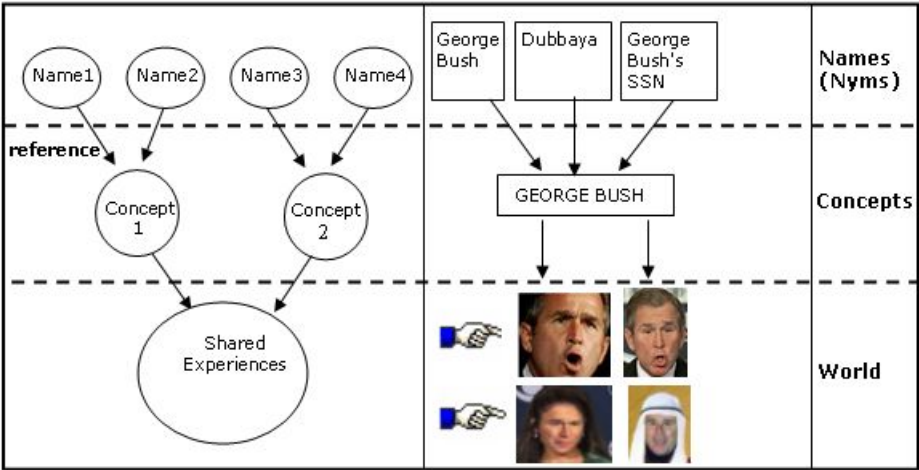


Fig. 1. Layers used in model

3 Model Semantics and Identical Relation

The crucial assumption here is the ability to define that two NAMES refer to the same CONCEPT. This “IDENTICAL” relation forms the core of our model. Of secondary importance (because this kind of identification is rare in digital environments) is the ability to determine that two concepts refer to the same real world entity. We call these two types of IDENTICAL – conceptually IDENTICAL and demonstratively IDENTICAL. They correspond to the two examples:

- 1. Conceptually Identical:“Is George Bush Dubbaya?” (Names refer to same concept.)
- 2. Demonstratively Identical:“Is that GEORGE BUSH [concept1], THE CURRENT U.S. PRESIDENT [concept2]?” Concepts refer to same real world experience via demonstrative pronoun.)

4 Specific Terms Used in This Model

4.1 Nym

Various terms are used for the elements in the NAME level of Fig. 1. For example RDF Model Theory uses “Literal” and “URIRef”. We choose the specific term NYM for the subset of NAMES used in identity management contexts. NYMs as opposed to NAMES have the following specific properties.

- NYMs are NAMES for which we are trying to determine relationships of identity.
- Generally a NYM only refers to a human being, an online “persona” or an utonomous software agent.

4.2 Fact

We take the meaning of FACT from knowledge representation as a component of a knowledge base (henceforth KB). A FACT is a simple well-formed statement, assumed to be true for a given system or KB [16]. We use the notion of a FACT as an atomic Subject-Predicate-Object statement as is used in RDF. For reasons of simplicity and to align our model with RDF syntax, it is convenient to normalize all facts to the form $\text{Fact}(\text{Nym}): \{(\text{Subject}) \rightarrow (\text{Predicate}) \rightarrow (\text{Object})\}$ E.g. $\text{F}(\text{John Smith}): (\text{John Smith}) (\text{has email address}) (\text{john.smith@smith.tv})$. I.e. we use the notation $\text{F}(\text{NYM})$ to denote a FACT as predicate, with a certain NYM as subject. Such facts are taken to have a definite truth value for each Nym. For example $\text{F}(\text{NYM1})$ has a different value from $\text{F}(\text{NYM2})$. E.g. John Smith is hated by Sandra might be true but Tracey Smith is hated by Sandra might not be true.

FACTs may also be composed using predicate logic to form compound facts.

We assume logical closure for sets of FACTs: if we know all passengers on plane x have SARS and we know Jon Smith is on plane x , then we know Jon Smith has SARS

5 Core Model

5.1 Identical Relation of 2 Nyms

We begin our model of the IDENTICAL relation of 2 NYMS with the principle of the Identity of Indiscernibles [17], which defines the relation IDENTICAL for two objects (NYMS in our model) as follows. *If, for every Fact $F(N1)$ is true, $F(N2)$ is also true, then $N1$ is identical to $N2$.*

$$N1 == N2 \text{ iff } (\forall F)(F(N1) \leftrightarrow F(N2)) \quad (\text{Lemma 1})$$

If and only if every property attributed to two NYMs is the same (including spatial co-ordinates etc...), then they refer to the same concept. One cannot determine all possible facts about two NYMs, but the more predicates are true of both, the more we are likely to believe they are identical. In practice, we generally use the method of identification by elimination, based on the inverse of Lemma 1, $N1 \neq N2$ iff $\exists F(F(N1) \neq F(N2))$ (5.3.7.) or we use the method of unique identifiers (5.3.8.).

5.2 Scenario 1. Mistaken Identity

In a murder in Livorno, Italy there was DNA identification process, which through Interpol was traced to a barman (Peter Neil Hamkin) in Liverpool, UK [19]. The barman claimed he had never been to Naples and it was then discovered that there was an error in the police DNA archives.

NYMs: Murderer, Barman

The following FACTs uniquely identify the barman as the murderer:

F1(Murderer): Murderer has a DNA print in file form as a hexadecimal string
DNAMurderer=578AF3..

F1(Barman): Barman has a DNA print in file form as a hexadecimal string
DNACain=578AF3..

F2: DNA prints are unique to living persons.

F3: Barman and Murderer are living persons

The following FACTs suggest $\text{Murderer} \neq \text{Barman}$ because $F4(\text{Barman})$ is true but $F4(\text{Murderer})$ is not:

$\neg$ **F4(Barman):** Barman was not in Naples on 5th of August 1994.

F4(Murderer): Murderer was in Naples on the 5th of August 1994

In fact it was proven that $\text{Murderer} \neq \text{Barman}$ because not all of the first set of FACTs were true

5.3 Identification, Linkability, and Anonymity with Respect to (wrt) an Anonymity Set

5.3.1 Anonymity Set. ANONYMITY SET is a set of NYMs wrt which we are trying to establish an identical relation for a given NYM. For example Persons, IP addresses etc... This term is taken from the work of [9]

5.3.2 Candidate Set. The set of NYMs in an ANONYMITY SET which *could be* identical to a NYM *given* a set of FACTs.

5.3.3 Identification. IDENTIFICATION for a given NYM, occurs when, based on a new set of FACTs (i.e. facts *not in the PRIOR KNOWLEDGE BASE* – see 5.2.10), the number of NYMs in the CANDIDATE SET decreases. The process of IDENTIFICATION is a narrowing of the set of candidates for identity.

Given the *PRIOR KB*, the CANDIDATE SET already has a well defined size (which may be equal to that of the ANONYMITY SET). IDENTIFICATION occurs when a set of new FACTs *reduces* the size of the CANDIDATE SET (e.g. using the inverse of Lemma 1.). In the case of UNIQUE IDENTIFICATION, the CANDIDATE SET is reduced to 1 member. N.B. the context dependence of identification upon 1. the *PRIOR KB* and 2. the ANONYMITY SET (see sect 5.3.)

The role of the *PRIOR KB* is illustrated as follows. Consider an *PRIOR KB* which is the single FACT: “email addresses are unique to each member of the ANONYMITY SET”. The knowledge that a NYM being identified and one of the ANONYMITY SET have the same email address is then uniquely identifying. But without this *PRIOR KB*, it is not identifying. If both the email address and the uniqueness condition are already known, no other fact can identify the NYM further.

5.3.4 Linkability. LINKABILITY is a measure, for a NYM and a set of FACTs, of the size of the CANDIDATE SET of NYMs relative to the size of the ANONYMITY SET (see Fig. 2). The larger the CANDIDATE SET, the less the LINKABILITY. Complete ANONYMITY (unobservability) is when the NYM being identified could

be IDENTICAL to *any* NYM in the ANONYMITY SET. Its opposite, UNIQUE IDENTIFICATION is when the NYM could only be IDENTICAL to one NYM in the ANONYMIT SET.

5.3.5 Anonymity. Anonymity is the opposite of LINKABILITY. In mathematical terms (see section 5.3.9) ANONYMITY = 1- LINKABILITY.

5.3.6 Pseudonymity. PSEUDONYMITY is the generic name for the quantity which decreases with LINKABILITY and increases with ANONYMITY.

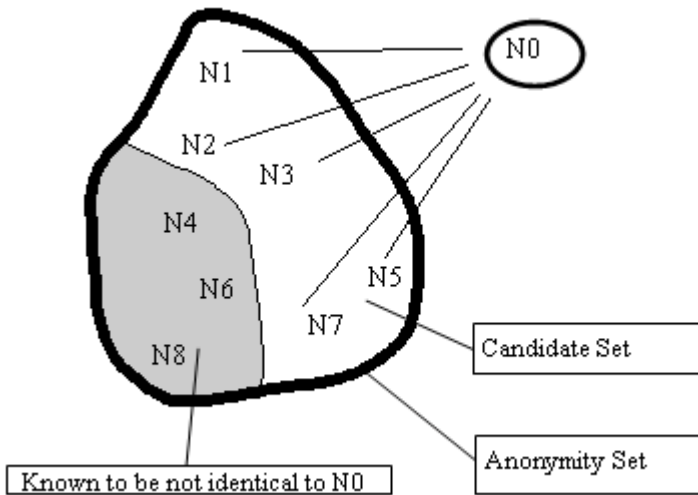


Fig. 2. Diagrammatic representation of Identification

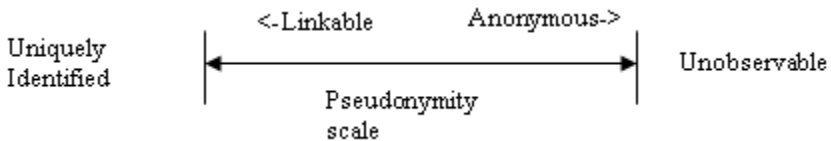


Fig. 3. Relationship between LINKABILITY and ANONYMITY

5.3.7 Identification by elimination. It is the inverse of Lemma 1. ($N1 \neq N2$ iff $\exists F (F(N1) \neq F(N2))$), which is much more often used: in words, if a fact is true of one NYM but not another, then they are NOT identical. In the real world, we often know that a particular NYM is identical to one of a set of NYMs. For example we might know for sure that a particular person was a male human being. Suppose we also know that people who buy Viagra are usually over 50. If we are targetting a mailing list of people, we can immediately eliminate anyone under the age of 50 from our mailshot.

5.3.8 Identification by Unique Identifiers. Certain facts are known to be true for only one NYM in a certain ANONYMITY SET. For example the FACT that a certain NYM has the same fingerprint file as one US CITIZEN (named by an SSN), identifies that NYM uniquely among US CITIZENS. If John Smith has fingerprint file (in hex format) 16b5c8 and the US CITIZEN referred to by SSN 12345 has fingerprint 16b5c8, John Smith is uniquely identified among the ANONYMITY SET of SSN's. Formally, if $UF(N)$ is a unique fact wrt an ANONYMITY SET A :

$$UF(N_0, A(N_1 \dots N_i)) \wedge UF(N_x, A(N_1 \dots N_i)) \rightarrow N_0 = N_x \quad (\text{Lemma 2})$$

If a FACT, known to be true for only one member of ANONYMITY SET $A(N_1 \dots N_i)$ is true of both N_0 (the NYM being identified) and N_x ($\in A$), then $N_0 = N_x$.

5.3.9 Prior Knowledge Base. The background KB, against which new facts identify a NYM. E.g. a fingerprint file can identify a PERSON uniquely given the PRIOR KB: the fact that finger print files are unique to PERSONs.

5.3.10 Scenario 2. Medical Records. The following simple scenario illustrates how this model of IDENTIFICATION, LINKABILITY and ANONYMITY corresponds to practical identification scenarios. Suppose we have a set of NYMs, which correspond to the subjects of a particular set of health records. These NYMs are a set of anonymous Health Record Numbers about which sets of associated FACTs such as blood groups and disease test results are known. Now suppose we want to know which of these NYMs might be identical to the NYM John Smith. According to the definition of IDENTIFICATION given above, all we need to do in order to uniquely identify John Smith with one of the NYMs is to find sets of FACTs, which are known to be true of only one individual. Thus the set of FACTs

F1(John Smith): John Smith has telephone number +391234567

F3(HRN1): HRN1 has had aids

F4(HRN2): HRN2 has not had aids

F5(HRN1): HRN1 has telephone number +391234567

F6(HRN2): HRN2 has telephone number +393413363

F7: Telephone numbers are unique to persons

F8(HRN3): HRN3 has telephone number +393413363 Etc...

identify John Smith uniquely as HRN2 in the the health records ANONYMITY SET, because it differentiates the NYM "John Smith" from all other NYMS in the set. These FACTs for this set of NYMS do not preserve ANONYMITY. They reduce the CANDIDATE SET to 1, whereas the set of FACTs,

F1: John Smith has an Italian phone number telephone number +39....

F3(HRN1): HRN1 has had aids

F4(HRN2): HRN2 has not had aids

F5(HRN1): HRN1 has an Italian telephone number +39.....

F6(HRN2): HRN2 has an Italian telephone number +39.....

F7: Telephone numbers are unique to persons

F8(HRN3): HRN3 has telephone number +39.....

F10(HRN4): HRN5 has telephone number +44..... Etc...

identify John Smith only weakly in the ANONYMITY SET of these health records because they differentiate the NYM “John Smith” only from HRN4 and reduce the CANDIDATE SET only by 1/5. The degree of LINKABILITY depends on the size of the ANONYMITY SET and the FACTs known.

5.3.11 Mathematical expression of Linkability and Anonymity. From this, we can derive a measure of LINKABILITY and its opposite, ANONYMITY. Suppose N_x is ANONYMOUS with respect to a set of NYMs A_i with I members (cardinality i) where K is a KB. Suppose further that $I(N_x, K, A)$ is the size of the candidate set of NYMS which *could be* identical to N_x in a given ANONYMITY SET A_i , *given* a set of facts, K then we have

$$\text{LINKABILITY } L = \frac{I(N_x, K, A_i)}{i} \quad (\text{Lemma 3})$$

And the ANONYMITY is

$$\text{ANONYMITY } A = 1 - \frac{I(N_x, K, A_i)}{i} \quad (\text{Lemma 4})$$

LINKABILITY is always between 0 and 1 and all other factors being equal, it decreases, the larger the size of ANONYMITY SET. The maximum value, 1 is reached with UNIQUE IDENTIFICATION. The converse is true of ANONYMITY.

5.4 Identities of a Nym wrt Anonymity Sets and Facts

We wish to define an IDENTITY of a NYM as any set of facts which *increases its linkability* wrt a given KB and ANONYMITY SET. If a set of facts differentiates a Nym from other NymS in its ANONYMITY SET, then it is part of its IDENTITY.

It is important to note that this makes IDENTITY *highly* context dependent. Some interesting consequences are

- Wrt some KB, any non-tautologous FACT may be part of a NYM's identity – see example at the end of this section.
- FACTs in the existing KB are not part of an IDENTITY.
- Wrt a PRIOR KB which already uniquely identifies a NYM, *no fact* is part of that NYM's IDENTITY.
- All other things being equal, the larger the ANONYMITY SET, the less identified a NYM is.
- Only if with a stable ANONYMITY SET and PRIOR KB are the classes of facts which increase an individual's linkability also stable.

Formally, an IDENTITY of a Nym $N1$ relative to an ANONYMITY SET A and a KB, K is any set of facts I defined by the following Lemma

$$I: I \neq \{\} \text{ AND } \forall (\hat{I} \subset I, \hat{I} \neq I) \ L(N1, (I \cup K), A) > L(N1, (\hat{I} \cup K), A) \quad (\text{Lemma 5})$$

Note that this implies $L(N1, (I \cup K), A) > L(N1, K, A)$. In words, I is any minimal, non-null set of FACTs which increases the LINKABILITY of the NYM wrt A and K . Minimal here has the sense that there is no smaller set of FACTs which has the same LINKABILITY. This final condition is important to our definition because otherwise any irrelevant FACT would also be part of an IDENTITY – it ensures that all the FACTs in the IDENTITY contribute to the process of IDENTIFICATION.

At this point, it is relevant to discuss the work of Kohntop [18], which distinguishes between the full and partial IDENTITIES of a NYM wrt an ANONYMITY SET. Intuitively it appears that an identity is some fixed set of FACTs, which can be said to identify a NYM. However, as we can see from the above definition, IDENTIFICATION is so much dependent on the identification context and the PRIOR KNOWLEDGE BASE, that actually *any* (non-tautologous) FACT may in some context be part of a given NYM's identity. In the light of this, it is very difficult to provide a useful definition of a full IDENTITY. Perhaps then, the term FULL IDENTITY is only useful as a kind of abstract container for the set of partial identities rather as the term infinity is used in the context of sets of numbers.

Furthermore it should be noted that the notion of an IDENTITY as a fixed set of FACTs is only useful in contexts where both the ANONYMITY SET and the assumed KB are also fixed and stable. A good example of this is the ANONYMITY SET of all people and the assumed KB that fingerprints are uniquely identifying. In this case it is reasonable to say that fingerprint data constitutes a partial IDENTITY.

The following example shows how facts which have no apparent connection with the data subject may identify him, with the following assumed KB,

F1: Mickey Mouse uses Prozac.
the facts (together)

F2: John Smith doesn't use anti-depressants, **F3:** Prozac is an anti-depressant.

are an IDENTITY of Mickey Mouse within the ANONYMITY SET of persons, even though neither directly concerns Mickey Mouse. Note that they are only part of Mickey Mouse's IDENTITY with F1 in the assumed KB. This shows the context dependence of identification wrt an existing KB.

6 Related Concepts

6.1 Personal Data

PERSONAL DATA is the class of facts whose subject is any NYM referring to a concept in the class of PERSON. PERSON is just a concept with an agreed range of reference. The reference includes objects with characteristics, such as thoughts, bodies, addresses, etc... but is of course the subject of a philosophical debate which we will not enter into in this paper. E.g. in a chat room, if "Bugs Bunny" is taken to refer to a real PERSON, then "Bugs Bunny has pneumonia" is personal data. If Bugs Bunny refers to a cartoon character, then "Bugs Bunny has pneumonia" is not personal data.

6.2 Personally Identifiable Data

To define the concept of personally identifiable data, we simply link the concept of PERSONAL DATA (6.1) with the concept of IDENTIFICATION (5.2.2.) to give the following definition: The definition follows the same form as Lemma 5., with the ANONYMITY SET defined as Persons.

Examples

1. “X is a person” is not PII because it does not increase the linkability of X within the set of persons.
2. F1: “Prozac is an anti-depressant” is not PII for the same reason.
3. But within the set of facts:

F1:Prozac is an anti-depressant

F2:Steve uses Prozac

F3:Robin Smith uses anti-depressants

F4:Cary Wilson doesn’t use anti-depressants etc...

{F2} and {F1 given the PRIOR KB (F3-4...)} constitute PII wrt Steve, but F1 and F3, F4 alone do not. As with non-personal identification, note that F1 is only part of STEVE's identity with F2-5 in the existing KB.

6.3 Pseudonym

The CONCEPT of a PSEUDONYM only has meaning relative to a set of NYMs when

- We have more than one identical NYM and
- One NYM is defined to have a special status with respect to others.

E.g. one might define firstname + lastname to be a person's “real name”. All other NYMs identical to that NYM are then its PSEUDONYMs. The term PSEUDONYM (from Gr. pseudonumos, falsely named) is used in a context in which the facts available are not sufficient to establish the identity of the NYM and PSEUDONYM within the ANONYMITY SET, thus allowing the PSEUDONYM to hide its identity. The use of PSEUDONYMs increases the level of PSEUDONYMITY.

6.4 Identity Theft

The PERSON referred to by a NYM N1 can steal the IDENTITY of a PERSON referred to by NYM N2 by falsifying facts such that given the fact set available about N1 and N2, they are deduced to be identical. To detect IDENTITY theft, one needs 2 knowledge bases - one in which the false IDENTITY is sustained, and the other in which it is not – consider for example:

F1(N1):N1 lives at via lungo lago 5

F2(N1):N1 has Italian passport number 123456789

F3(N1):N1 has SSN 012387

F2(N2):N2 has Italian passport number 123456789

F3(N2):N2 has SSN 012387

With this set of FACTs, we can sustain that N1 and N2 are IDENTICAL because Lemma 1. ($N1 = N2$ iff $(\forall F)(F(N1) \leftrightarrow F(N2))$) is satisfied. Whereas for

F1(N1): N1 lives at via lungo lago 5
F2(N1): N1 has Italian passport number 123456789
F3(N1): N1 has SSN 012387
F2(N2): N2 has Italian passport number 123456789
F3(N2): has SSN 012387
F4(N2): lives in Via Enrico Fermi 1
 Implication: $(F4(N2) \rightarrow \neg F1(N2))$

We cannot claim $N1=N2$ because $F(N1) \neq F(N2) \rightarrow \neg (\forall F)(F(N1) \leftrightarrow F(N2))$.

6.5 Credential

The classical model of authorisation grants access to services through usernames and passwords. FACTs linked to a username and password verify whether the user is authorized. Often however, this data is not actually required to carry out the service. Its request can therefore contradict the principle of data minimization set out in the EU Data Protection Directive [2] article 6, whereby data should be “adequate, relevant and not excessive in relation to the purposes”. Therefore recent developments in privacy enhancing technologies have emphasized the use of credentials, which are certified, minimal facts required to use a service. E.g. to buy duty free goods in an EU airport shop, the required information is an assurance that I am a bona fide passenger travelling from the EU to the US, not as is current requested all the data contained in a boarding card. In terms of our model, a credential is minimum facts, certified by some trusted means, for the provision of a given service.

6.6 Relationship to Database Models

There is a clear analogy between this model of IDENTITY and database structure. In terms of relational databases, a NYM corresponds to a key and a fact to a field value for a key. Facts of type F2 above correspond to foreign keys. In the object oriented database model, a NYM would refer to an object name and a fact to an object, attribute, value triple. FACTs of type F2 relate to the NYM which is the subject of the identity via inheritance.

7 Relation to Formal Ontology Languages

The main problems in translating such a model to formal syntax are in deciding which parts are actually *already part* of the formal model of RDF, RDFS etc... The use of ontology editing tools is also difficult as they do not allow mixing of model concepts with those of the base model. The precise determination of which concepts should be shared requires further study but we have made some suggestions in Fig. 5. We suggest for example, that the RDFS concept of a CLASS may be equivalent to an

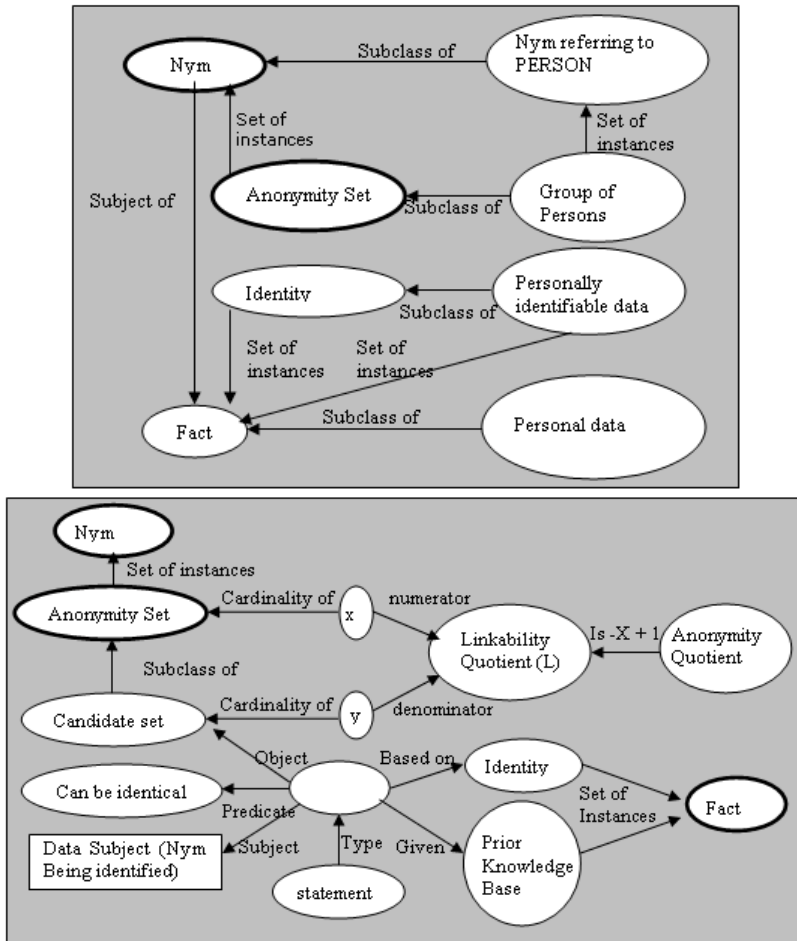


Fig. 4. Abstract RDF graph, with no assumption of concept sharing with RDF

ANONYMITY SET? More tentative suggestions are the possibility of equating facts with RDF statement and NYMs with URIRefs. In further work, these graphs will be used as a basis of a formal ontology expressed in a syntax such as OWL.

8 Conclusion

Digital identity can be seen as being established by determining overlaps and contradictions between FACTs about NYMs (names/identifiers) referring to concepts (e.g. of a person) and not directly to real world objects. This explains the growing perception of a need for legislation which is able to defend identities based on facts in databases and not on a confused notion of “real world” objects, a notion which becomes less and less coherent as identities become more abstracted. The above

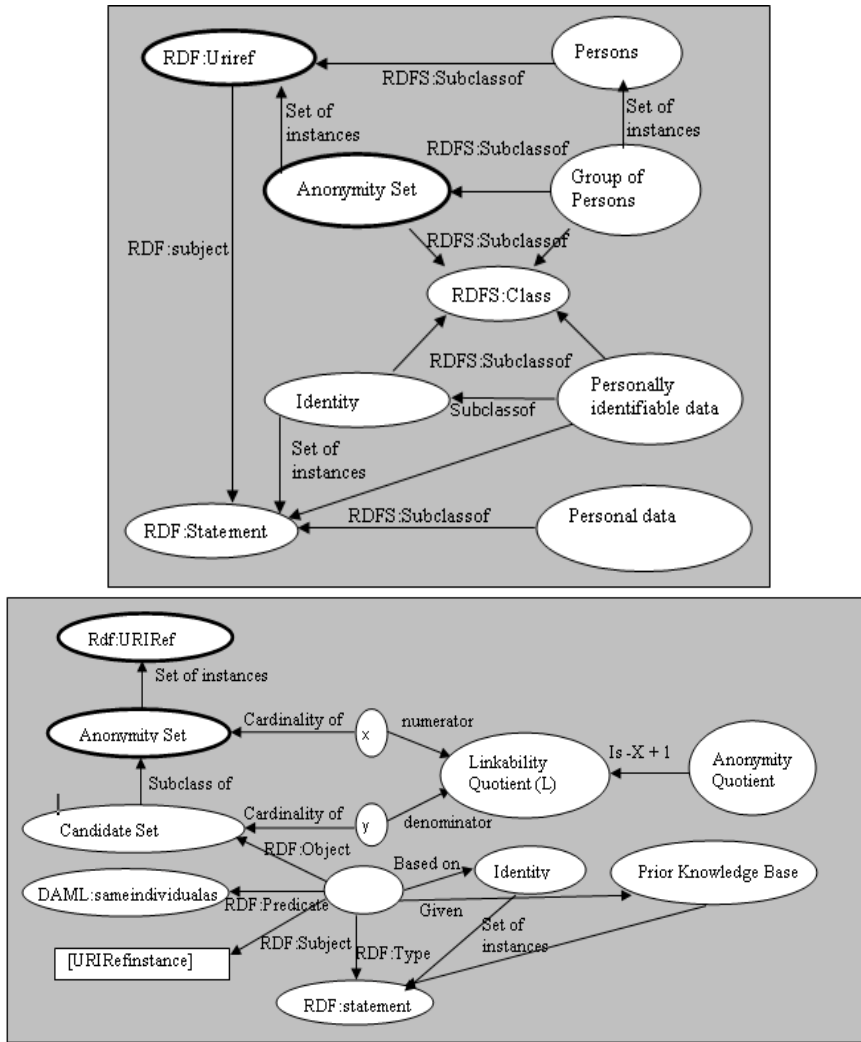


Fig. 5. Preliminary model using RDF, RDFS and OWL

model describes the processes of identification in purely information theoretical terms which eliminate many contradictions and circularities of previous models.

We assume a perceived world of distinct objects, but we do not confine ourselves to the notion of unique identification of these objects or the concepts which refer to them. Identification is an incremental process of distinguishing smaller and smaller groups of NYMs in the context of an ANONYMITY SET and a PRIOR KNOWLEDGE BASE. This model highlights the context dependence of identification and the urgent need to include this aspect in legal and technical models.

Acknowledgements. Thanks to Neil Mitchison and Marcelo Masera for comments.

References

- [1] US Patriot Act. (Oct.25,2001)
- [2] European Parliament and Council: Directive 95/46/EC, The protection of individuals with regard to the processing of personal data and on the free movement of such data (1995)
- [3] Art 29 working document: Privacy on the Internet – an integrated EU approach to on-line data protection. WP 37, 5063/00/EN/final (2000)
- [4] DARPA Agent Markup Language - see <http://www.daml.org/>
- [5] RDFS: W3C Working Draft:<http://www.w3.org/TR/2003/WD-rdf-schema-20030123/>
- [6] OWL: W3C Working Draft: <http://www.w3.org/TR/owl-semantic/>
- [7] Hogben,G. (ed.): JRC white paper on the use of ontologies for PETs
<http://pronto.jrc.it/files/bestpractice.doc>
- [8] Clauß, S. Kohntopp,M.: "IDENTITY management and its support of multilateral security" Computer Networks 37 (2001) 205–219
- [9] Pfitzmann,A., Kohntopp,M.: "Anonymity, Unobservability, and Pseudonymity – A Proposal for Terminology" v0.14 (2003), <http://www.cs.rmit.edu.au/fedconf/home.html>
- [10] Clarke,R.: Human Identification in Information Systems: Management Challenges and Public Policy Issues (1994)
<http://www.anu.edu.au/people/Roger.Clarke/DV/HumanID.html>
- [11] Clarke,R.: "Identified, ANONYMOUS and Pseudonymous Transactions: The Spectrum of Choice" (1999) <http://www.anu.edu.au/people/Roger.Clarke/DV/UIPP99EA.html>
- [12] W3C Standard: Resource Description Framework, <http://www.w3.org/TR/rdf-mt>
- [13] Johnson-Laird, P. N., Mental models in cognitive science. Cognitive Science, 4, (1980).
- [14] Fodor, J.: The Language of Thought. New York: Crowell., CONCEPTS: WHERE COGNITIVE SCIENCE WENT WRONG. By Jerry A. Fodor. New York: OUP (1998)
- [15] Chomsky, N.: Knowledge of Language, NY: Praeger (1986)
- [16] Russell,S., Norvig,P.: "Artificial Intelligence: A Modern Approach" Prentice Hall (1995)
- [17] Loemaker (ed.): Wilhelm Gottfried Leibniz, Discourse on Metaphysics, Section 9 (1969)
- [18] Hansen,M., Berlich,P.: Identity Management Systems: EMTEL Proceedings (2003)
- [19] See for example <http://www.repubblica.it/online/cronaca/livorn/killer/killer.html>

Ontology-Based Customer Complaint Management

Mustafa Jarrar, Ruben Verlinden, and Robert Meersman

STARLab - Vrije Universiteit Brussel,
Pleinlaan 2, Brussels, 1050, Belgium
{mjarrar, rverlind, meersman}@vub.ac.be
<http://www.starlab.vub.ac.be>

Abstract. This paper presents an ontology-based approach for managing and maintaining multilingual online customer complaints. To achieve trust and transparency in e-commerce communications and transactions, effective and cross-border complaint platforms need to be established and may be integrated in e-business activities. The effectiveness and width of such complaint service platforms depend on rising to several challenges, such as the sensitivity of business regulations and complaint resolution, the language and cultural diversity of the cross-border business parties, the extensibility according to the market needs and standards. In this paper, we show how such challenges can be addressed and simplified: first, we propose the construction of an ontology that captures the core knowledge of the customer complaint domain. Second, we show how the extensibility of a complaint platform can be simplified and managed. Finally, we show how a multilingual representation of this ontology may be constructed.

This paper outlines our main achievements in Topic Panel 6 (“Ontology, Extensibility and Integration”), which is a special interest group in the EU CCFORM Thematic Network project¹.

Keywords: Customer Complaint Management, CRM, e-CRM, Ontology, Core Ontology, Customer Complaint Ontology, DOGMA, ORM, Multilingual Representation of Ontologies.

1 Introduction and Background

The use of the Internet for cross-border business is growing rapidly. However, in many cases the benefits of electronic commerce is not exploited fully by customers because of the frequent lack of trust and confidence in online cross-border purchases. To achieve fair trading and transparency in commercial communications and transactions, effective cross-border complaint platforms need to be established and involved in e-business activities [CIHF02] [CW87].

The CCFORM project aims to study and *reach consensus* about the foundation of online customer complaint management mechanisms by developing a standard but extensible form (called CC-form²) which has widespread industry and customer

¹ (IST-2001-34908), 5th framework.

² We refer to the project as CCFORM and to the target customer complaint form as “the CC-form”.

support. This CC-form must facilitate cross-language communication to support cross-border e-commerce and should be easy to implement in software tools. The CC-form will raise the basic standard of complaints management, and can be extended in vertical markets to provide sector-wide solutions, and by service providers to gain competitive advantages.

The main challenges of establishing and standardizing such a CC-form are: (1) the legal bases: the sensitivity on cross-border business regulations and privacy issues, (2) the diversity of language and cultural aspects: controlling and standardizing the semantics of the complaint terminology i.e. the intended meaning of the terms, and among different human languages, (3) customer sensitivity and business perspectives, (4) extensibility: the flexibility of extending the CC-form according to market needs and standards, e.g. extending the kinds of problems that a complainant can complain about, extending the kinds of resolutions, managing who may extend what, etc.

In order to tackle such challenges and to perfect the reference model for the complaint form, i.e. CC-form, the major work in the CCFORM project has been divided into six topic panels, each consisting out of 10-15 specialized members. Each panel has been intensively discussing different issues: TP1: Legal Affairs, TP2: Consumer Affairs, TP4: Standards for SMEs. TP5: Alternative Dispute Resolution Systems, TP6: Ontology, Extensibility and Integration, TP7: Vertical markets.

This work outlines our main achievements in the “Ontology, extensibility and integration, including multilingual and cultural issues” topic panel. The goal of this topic panel is to undertake the extensibility and multilingual demands. To approach this, a customer complaint ontology (CContology), lexicalized in multiple languages has been developed.

In the next section, the CContology will be presented. In section 3 we discuss the extensibility and organization of the CContology. In section 4 we show how an ontology can be lexicalized in different human languages. Finally, in section 5 some conclusions are drawn.

2 Customer Complaint Ontology

The *customer complaint ontology* (CContology) intends to capture the knowledge elements (present in a so-called *conceptualization*) of the “customer complaint management” domain. Its core covers a semantic description of complaints that could be issued by any legal person against any other legal person (NGO, company, natural person, government, etc.). It comprises business models, categories of complaints and resolutions, “best-practice” business rules, etc.

The main intended impact of the CCFORM project is that the results of the project will be a future initiation of a European online complaint platform that will provide a trusted portal between consumers and business entities. In this respect, the ontology is intended to become the basis for a future so-called *core ontology* in the domain of customer complaint management. Applying the CContology in such an European online complaint platform will enable the further refinement of the CContology.

The main use of such an ontology is 1) to be shared among all stakeholders and thus enable consistent implementation (and interoperation) of all their software complaint management mechanisms based on the shared background vocabulary; and 2) the CContology holds *business rules*, defined between concepts and relations, so

that that *valid information structures can be enforced*. Furthermore, 3) to play the role of *core domain ontology*; it encompasses the core complaining elements, and *can be extended by individual or groups of firms*.

Although this CContology has been built and approved by all of six topic panels, in its current state it can only be considered a proposal. The CCFORM community, while representative for a sizable cross-section of the domain, is not a standardization body nor in a position for a *de facto* enforcement of this ontology as a generally agreed semantic specification. However, we claim the approach presented in this paper is designed to initiate and drive such a process.

The CContology is modularized into a set of (at this moment seven) so-called ontological commitment modules³: Complaint, Complainant, Recipient, Address, Complaint Problems, Complaint Resolutions, and Contract. Each commitment module consists of a set of binary conceptual relations and rules that specify and constrain the intended meaning of the concepts and relations. *An inclusion composition relationship can be defined between the commitment modules: all concepts and rules introduced in the included commitment module will be inherited in the including commitment module.*

Any complaint form, including the completed forms (i.e. the data), should be based on (i.e. commit to) the semantics represented in the CContology. Formally, for an application to commit to a certain commitment, it must satisfy all rules declared in this commitment. In other words, any possible world, for an application, must conform to the rules declared in its commitment(s) (cf. model-theoretic semantics). In [JLVM03], we illustrated an approach for semi-automatic ontology-driven generation of web forms: to model a CC web form, one selects the appropriate commitment modules, composes them and then automatically generates the target web form from these composed commitment modules.

Let us now turn to present the content of the ontological commitment modules. The CContology is developed using the DogmaModeler ontology engineering tool [JDM03]. The commitment modules are represented graphically in figures 1 to 7 using ORM [H01], a conceptual graphical modeling notation⁴ that is used in

³ Notice that the commitment modules are not the ontology itself. An ontology, in our approach [JDM03][JM02], consists of an *Ontology Base* (holds intuitive domain knowledge) and a set of *ontological commitments* (each holding *formal* application-kind knowledge). Due to readability issues for non ontology-experts we have introduced the commitment modules before introducing the ontology base, which is presented in section 4.

⁴ In ORM, ellipses denote concepts (e.g. “complaint”) and rectangles denote relations. Each relation consists of two roles (e.g. “describes” and “described by”). Rules in ORM can also be represented graphically: the mandatory rule “●” between a concept and a role denotes that it is mandatory for the concept to play this role. The uniqueness rule “↔” on top of a role denotes that the concept can play this role only once. If the uniqueness rule spans over two roles, then it denotes that the combination of the two roles should be played at most once. The exclusive rule “⊗” between two (or more) subtype relations (see fig. 2) denotes that there is no intersection between the instances of the sub concepts. The Total rule “⊕” between two (or more) subtype relations denotes that the instances of the super concept are exactly the union of the instances of the sub concepts. The exclusive-or rule “⊕” between two (or more) *roles* (see fig. 3) denotes that the concept which plays these roles must play at least one of them. See [H01] for more about the ORM notation.

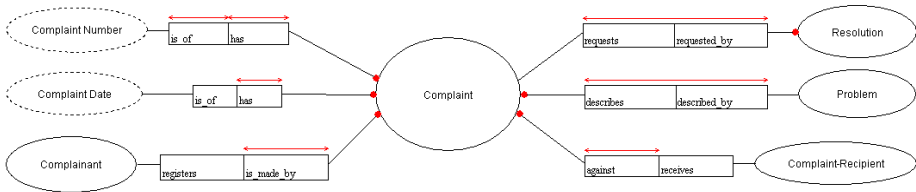


Fig. 1. The “Complaint” ontological commitment module

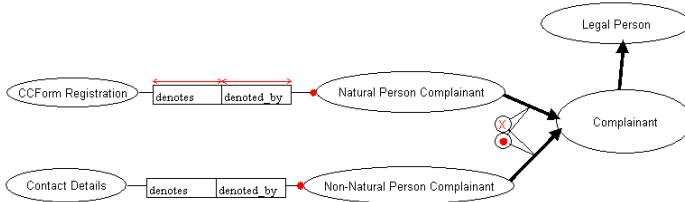


Fig. 2. The “Complainant” ontological commitment module

DogmaModeler. We provide an informal brief description of each commitment module after the corresponding ORM diagram. For the *formal specification* of the commitment modules, some prior background in ORM is required (e.g. see [H01]). The informal definitions of the concepts found in these commitment modules are provided in the “Customer Complaint Glossary”⁵.

Figure 1 illustrates the “Complaint” commitment module. A ‘Complaint’ is made by a ‘Complainant’ against a ‘Complaint-Recipient’ on a certain ‘Date’. A ‘Complaint’ is identified by its ‘Complaint Number’. In a ‘Complaint’ the ‘Complainant’ issues at least one ‘Problem’ to the ‘Complaint Recipient’ and he may ask for some ‘Resolutions’.

Figure 2 illustrates the “Complainant” commitment module. All ‘Complainants’ are ‘Legal Persons’. A ‘Legal Person’, as defined in the Glossary⁶, is any legal entity that has legal rights and responsibilities (e.g. it can be a Natural Person, a Company, a Government, an NGO, etc.). In the complaining context, and as commonly understood in most consumer regulations, a complainant can either be a consumer or a business customer, each implying a different legal basis for complaint handling. In Fig. 2 a complainant must be either a ‘Consumer Complainant’ or a ‘Non-Consumer Complainant’. Each ‘Consumer Complainant’ must have ‘Contact Details’. As will be shown in Fig.4, the mandatory contact information is only the ‘eMail’ and the ‘Country’ of the ‘Complainant’. A ‘Non-Consumer Complainant’ must have a ‘CCForm Registration’, which is an enrollment in the complaint platform that uniquely identifies the ‘Non-Consumer Complainant’ (and its various roles and contact details). The distinction between consumer and non-consumer complainants is not only based on the variation of their complaint handling regulations, but also on the preference of not obligating the inquiring of private information about consumers, such as his/her ‘Name’, ‘Mailing Address’, ‘Telephone’, etc.

⁵ The “Consumer Complaint Glossary” is not included in this paper, but it is publicly available at <http://www.starlab.vub.ac.be/ccform/CCGlossary.pdf>

⁶ See footnote 5.

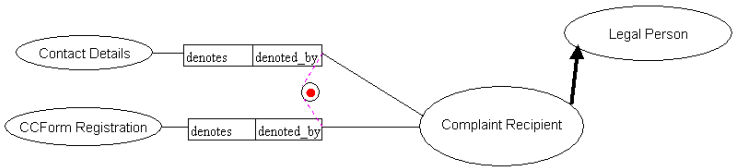


Fig. 3. The “Recipient” ontological commitment module

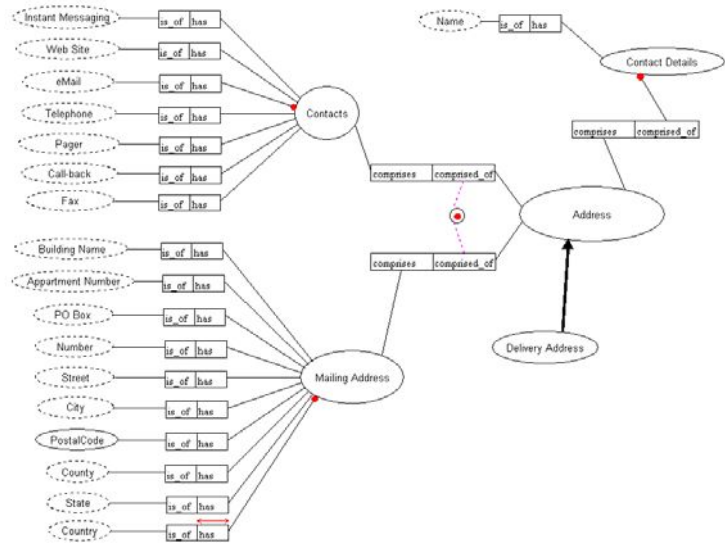


Fig. 4. The “Address” ontological commitment module

Figure 3 illustrates the “Recipient” commitment module. A ‘Recipient’ is a ‘Legal Person’. When a ‘Complaint’ is issued against a ‘Recipient’ the ‘CCform Registration’ or the ‘Contact Details’ need to be provided.

Figure 4 illustrates the “Address” commitment module. The ‘Contact Details’ consist of a ‘Name’ and an ‘Address’. An ‘Address’ is comprised of ‘Contacts’ and a ‘Mailing Address’. A ‘Mailing Address’ must have a ‘Country’, and can have all the traditional information of postal addresses in the European Union. ‘Contacts’ are used for non-postal communication and must include at least an ‘eMail’. In addition, ‘Contacts’ can include other non-postal communications ranging from ‘Fax’ to ‘Instant Messaging’.

Figure 5 shows the “Complaint Problems” commitment module. The content of this commitment is based mainly on the research that has been carried out by some of the other topic panel members, reported in [VS03]. A ‘Complaint Problem’ can be a ‘Privacy Problem’, or either a ‘Contract Problem’ or a ‘Non-contract Problem’. ‘Contract Problems’ maybe ‘Purchase Phase Problems’, ‘Pre-purchase Phase Problems’ or ‘Post-purchase Phase Problems’. If a problem is a ‘Purchase Phase Problem’ or a ‘Post-purchase Phase Problem’ the necessary information about the

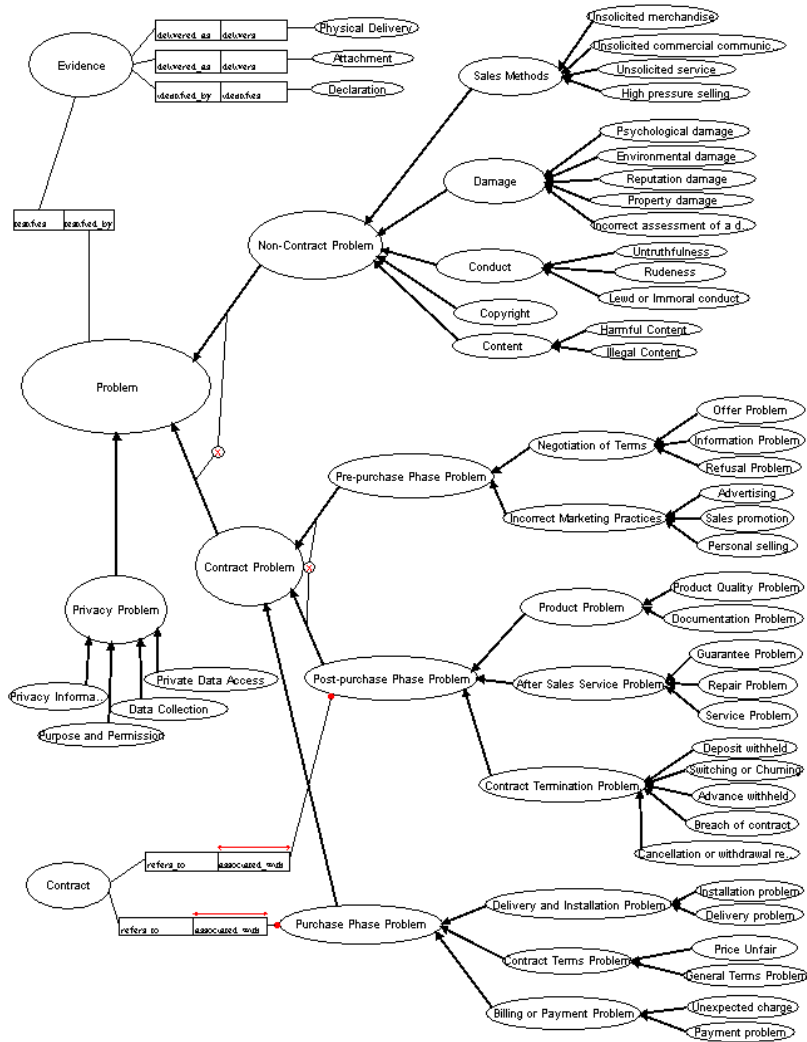


Fig. 5. The “Complaint Problems” ontological commitment module

‘Contract’ should be provided. ‘Complainants’ who register a ‘Complaint’ can optionally provide some ‘Evidence’ of the reported ‘Problem’.

Figure 6 illustrates the “Contract” commitment module. ‘Contract’ is a wider concept than “a piece of paper with on it a written agreement signed by two parties”. In CCFORM we speak of a ‘Contract’ from the moment there is a ‘Contract Order Date’ provided. Extra information about the ‘Contract’, such as ‘Reference to Contract’, ‘Contract Effective Date’ (Start date from which the contract is applied), the ‘Sales Office’, ‘Terms and Conditions’, etc. can also be provided.

Figure 7 illustrates the “Resolution” commitment module. A ‘Resolution’ requested by a ‘Complainant’ can be an ‘Economic Request’, a ‘Symbolic Request’ or an ‘Information Correction Request’. Etc.

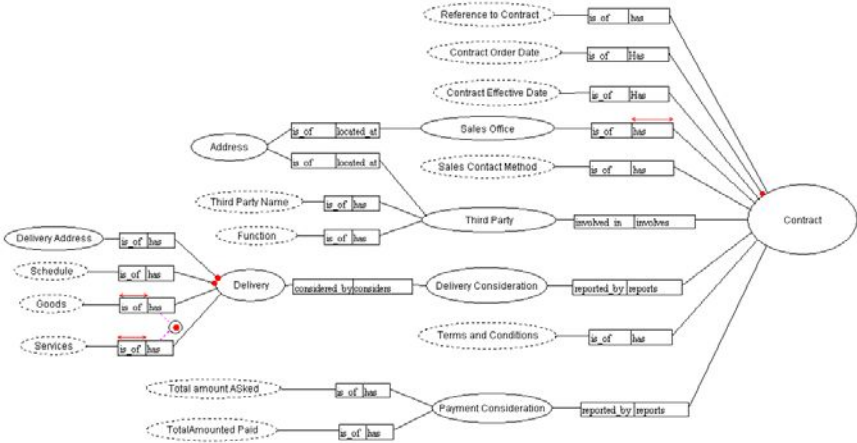


Fig. 6. The “Contract” ontological commitment module

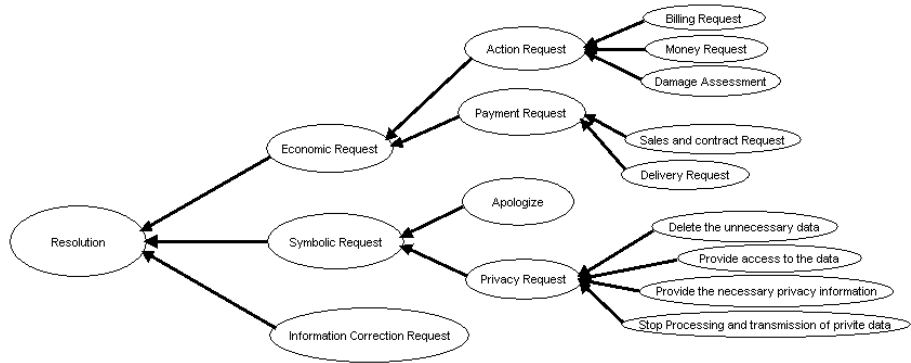


Fig. 7. The “Resolution” ontological commitment module

3 Modularization and Extensibility

Modularizing the ontology into a set of commitment modules, such as above, leads to extensibility, one of the main requirements for CC-form. Such extensibility of CC-form content may be required and performed by individual CC-form client companies. Once the CC-form is implemented as a centralized complaint portal between customers and companies, client companies may wish to extend “their” CC-form to inquire more specific complaint details, e.g. delivery conditions, product attributes, or they might wish to offer the customer a particular resolution, etc. Such extensions may be a necessity not only for individual companies but also in so called vertical markets applications (covered in the “vertical market” topic panel, TP7). In the CCFORM project, one of the main goals is to provide companies a method to extend the CC-form content themselves, within given (e.g. legal) constraints on those extensions. On the one hand, this will help to achieve a wider adoption of complaint

mechanisms in e-commerce applications, but on the other hand this will create new challenges: keeping the new extensions consistent with the existing CC-form and preventing misuse of the CC-form. For example, a company might try to misuse the CC-form by inquiring private information which violates the privacy regulations, or it may introduce new terminology and rules that are semantically inconsistent with the existing content terminology and rules.

In our solution the CC-form must not be altered directly; instead extensions are introduced into the CContology, the base of CC-form. Moreover, our modularization of the ontology -into a set on the ontological commitment modules- offers simplified methodologies for extending, maintaining and managing the CContology :

- *Extensions will not be allowed on all commitment modules.* For example, the “Complainant” and “Address” commitment modules may be locked, so companies will be prevented from e.g. asking privacy-rule-violating questions. Or perhaps, we can only allow extensions to be made into the “Problem” and “Resolution” commitments. In this way, we can achieve a “relatively” systematic management of the kinds of extensions allowed.
- *Extensions can be made and treated as separate modules.* If a company wishes to extend a certain commitment to inquire details about e.g. a certain kind of product, a new ontological commitment can be constructed to capture these details. In addition to that an inclusion composition relationship can be declared between the target commitment modules and the new commitment module.
- *Efficient maintenance and management.* CC-form may need to manage a large amount of extensions that target many dimensions of the CContology. Modularizing these extensions will make managing, maintaining and indexing them more scalable.
- *The development of the modules can be distributed among ontology experts, domain experts and application-oriented experts.* In the case of e.g. a vertical market application, where one wishes to develop a set of extensions (i.e. modules), the development and the review processes could be distributed according to the expertise of the developers and the subject of the modules.

For example, during the development of the core CContology we have distributed the development and review of the set of commitment modules to the specialized topic panels. Bistra Vassilev acted as domain expert for the development of the Problem and Resolution commitments even though she was based at several thousand kilometers distance. Members from TP1 (legal affairs) have reviewed the “Complaint”, “Complainant”, “Recipient”, “Address” and “Contract” commitments. Members from TP2 “Consumer affairs” have reviewed the “Complaint”, “Complainant”, “Problem” and “Resolution” commitments. Reviewing the CCglossary⁷ has been done by several members, etc.

- *Reusability issues.* One may wish to reuse some of the commitment modules in a broader context than the domains of complaints and complaint management. For example, the ‘Address’ commitment can easily be reused for tasks in other domains: Mailing, Marketing, Sales Force Automation, etc. The ‘Complaint Problems’ commitment module in the domains of market analysis, qualitative statistics, etc.

⁷ See footnote 5.

4 Multilingual Lexicalization of the CContology

This section proposes a methodology for lexicalizing the CContology (as obtained so far) in several natural languages. This methodology may be useful when one wishes to provide “multilingual” extensions to the CContology, or wants to translate the CContology “terms” into other natural languages. Lexicalizing the CContology into several natural languages supports the development of a software platform providing cross-language customer complaint form management. A multilingual lexicalization of the CContology assists a consistent and systematic translation of the terms expressing the concepts in the ontology. For complaint platforms, this allows the systematic translation of all elements of the generated and filled in customer complaint forms that do not contain “free” text.

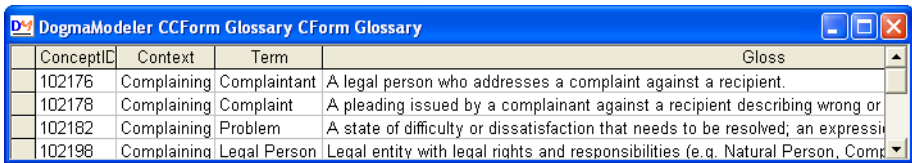
Terms in human languages in general can be ambiguous in that a term can have two or more distinct meanings. For example the term “bank” in English can have several meanings: a financial institution, sloping land (beside a body of water), etc. When humans communicate they usually have the capability and intuition to disambiguate the intended meanings of terms depending on the context in which these terms are used. However, ontologies are being used -in computing technology- to represent knowledge formally at the conceptual level; so that by sharing a conceptualization (i.e. an ontology) computer programs can interoperate meaningfully. See e.g. [G95][GG95] for further details about what is an ontology. In short, ontologies are intended to represent concepts rather than terms, implying a certain level of independence from natural language.

Before we introduce our methodology, we need to present some further basic principles of the DOGMA ontology engineering approach ([JM02a] [JDM03]) that we have adopted in this paper. In the DOGMA approach, ontologies are structured into an *ontology base* and a layer of *ontological commitments*. When developing an ontology according to DOGMA, the ontology base is constructed first, *then* a set of ontological commitments can be defined literally “in terms of” this ontology base. *Notice that for simplicity of presentation in this paper, we have presented the set of CC ontological commitments in section 2 before formally introducing its ontology base.* In short, an ontology base consists of context-specific binary conceptual relations, called lexons. In the CContology, only one context has been declared: “Complaining”. Within this context, the set of all complaining conceptual relations are defined. Table 1 shows some lexons of the CContology base.

The CContology commitment modules –presented in section 2– are defined within the “Complaining” context: the set of conceptual relations in each commitment are lexons in the “Complaining” context in the CContology base. In other words, the lexons and the intended meanings of their terms are shared among the set of ontological commitments. In DOGMA, each Term within a Context refers to a Concept. For example, since both the “Complaint” and the “Complainant” commitment modules are defined within the same “Complaining” context, the term ‘Complainant’ refers to the same concept in both commitments. By doing so, not only the modularization of the ontology is enabled, but also the “semantically safe” composition of a set of commitment modules is made possible [JM02b] through the inclusion interrelationship. Notice that when computer programs interoperate, they do not directly access and share the ontology base lexons, but they share and commit to the

Table 1. Example lexons of the CContology's ontology base

Context	Term _i	Relation	Term _j
Complaining	Complaint	has / is of	Compliant Number
Complaining	Complaint	is_made_by / registers	Complainant
Complaining	Complaint	describes / described_by	Problem
...	...	...	...
Complaining	Complainant	subtype_of / supertype_of	Legal Person
...		...	...
Complaining	Problem	supertype_of / subtype_of	Privacy Problem
Complaining	Problem	upertype_of / subtype_of	Contract Problem
...	...	...	...



ConceptID	Context	Term	Gloss
102176	Complaining	Complainant	A legal person who addresses a complaint against a recipient.
102178	Complaining	Complaint	A pleading issued by a complainant against a recipient describing wrong or
102182	Complaining	Problem	A state of difficulty or dissatisfaction that needs to be resolved; an expressi
102198	Complaining	Legal Person	Legal entity with legal rights and responsibilities (e.g. Natural Person, Com

Fig. 8. Example glosses of the concepts that are used in the CContology

intended meaning of these lexons through ontological commitments, which are formal knowledge (i.e. logical theories).

In the ontology base, each term within a given context⁸ (i.e. each concept) should have an informal description; this description⁹ –also called gloss– should provide a sufficient explanation for humans (who understand the specific language) to intuitively understand its intended meaning. Fig. 8 shows an example of glosses in English, defined within the CContology. “ConceptID” uniquely identifies a concept.

Accordingly, expressing concepts by terms within different natural languages is made easier *through the use of contexts*. Hence, we can translate terms (which are the lexical representation of concepts) in language A into terms in language B within a given context. For example, within the “Complaining” context, the English term “Complaint” can be translated into “Klacht” in Dutch or “Réclamation” in French. Notice that the term “Complaint” in English can have different other meanings in other contexts, e.g. “a loud cry (or repeated cries) of pain or rage or sorrow”, or “the first pleading of the plaintiff setting out the facts on which the claim for relief is based” in the context of civil law. In short, we seek *conceptual equivalence translations*.

Table 2 shows English-Dutch-French conceptual equivalence translations within the “Complaining” context of the terms presented in Fig 8.

To harmonize the different translations of the ontology our approach requires *an ontology to be build and lexicalized completely in at least one language*, the ontology’s *native language*. In the case of the CContology, English is chosen to be

⁸ More about the notion of context in the DOGMA approach can be found in [JDM03].

⁹ For documentation purposes.

Table 2. English-Dutch-French conceptual equivalence translation within the “Complaining” context

ConceptID	Context	English (Native)	Dutch	French
102176	Complaining	Complainant	Klager	Plaignant
102178	Complaining	Complaint	Klacht	Réclamation
102179	Complaining	Recipient	Ontvanger	Destinataire
102181	Complaining	Complaint Number	Klachtnummer	Numéro de Réclamation
102198	Complaining	Legal Person	Rechtspersoon	Personne Morale
..	..	..	..	..

Fig. 9. Simplified multilingual (English-French-Dutch) form

the native language. This native language then acts as *the* reference for translating the ontology into other languages.

Our conceptual equivalence translation approach yields sometimes imperfect translations but is scalable and easy to use. The CC-form can easily switch between different natural languages by substituting the terms with their conceptual equivalence translations from the ontology (assuming the contexts are sufficiently well circumscribed, of course). Figure 9 shows a simplified complaint form in English, Dutch and French.

While it is a pragmatic approach, the conceptual equivalence translation is not as trivial as it appears. Acquiring more elegant translations demands the translator to perform further investigation. In what follows, we present some issues and guidelines towards further convenience and high accuracy in the multilingual lexicalization of ontologies:

- *Cultural issues.* There is a high dependency between the language of people and their culture (social activities, religion, region, weather, interests, era etc.) Thus, within a community of people speaking the same language we can find different usage of terms, even within the same context. For example, within the “Complaining” context, when translating the term “Complaint” into Arabic there are two conceptually equivalent terms: “Mathaleem” and “Shakaoa”. In Palestine, the most commonly used term is “Shakaoa”, while in Saudi Arabia people prefer the term “Mathaleem”. Seemingly, the ideal solution for such a problem is providing a set of rules for the usage of each term, considering *all*

cultural issues [C98]. However, this does not yield a scalable approach for our purposes. Thus we advise that if such cultural variations are important for a context, it is better to treat e.g. English-UK, English-USA, Dutch-Belgium, Dutch-Netherlands, Old-Arabic, Modern-Arabic, etc. as distinct languages.

- *Word to word translation is not our goal.* Usually, the purpose of building an ontology is to formally represent an agreed conceptualization of a certain domain, and share it among a community of users. Thus, lexicalizing the concepts in an ontology into multiple languages is a manner of maximizing the *usability* of this ontology¹⁰, and not to play the role of multilingual lexicon. In lexicons or dictionaries, the purpose is to list only the common words –e.g. based on corpus– of a language with a description and lexical information. In ontologies it is normal to find a concept lexicalized by an expression. For example, “Total Amount Paid”, “Trying to obtain data improperly”, etc. Such concepts cannot in general be lexicalized in one word, in this case at least not in English.

To conclude, the methodology we have presented in this paper intends to maximize the usability of an ontology among several cross-language applications. Obviously, this methodology is useful and easily applicable in information systems that comprise forms, databases, XML and RDF tags, etc. Our methodology is however not suited in case of ontology-based natural language processing applications. For such applications we suggest the development of *multilingual ontologies*: developing formal representation (i.e. an ontology) for each human language considering all its concepts, and then a alignment layer to map between these ontologies. Further details on how to develop multilingual ontologies will be discussed in future papers.

5 Conclusion

In this paper we have presented our experience and main achievements in the Ontology, Extensibility and Integration topic panel, a special interest group in the EU Thematic Network project CCFORM.

Using ontologies as a foundation for cross-border online complaint management platforms can greatly improve the effectiveness, scope and extensibility of such platforms. While offering individual companies and organizations, or associations of them, advanced customization abilities by extension capabilities for the ontology, semantic consistency through the complaint management terminology is maintained. Furthermore, by restricting extensions to certain parts of the ontology, some legal constraints such as privacy regulations may be enforced systematically.

The proposed methodology for the multilingual lexicalization of ontologies is a pragmatic one. It offers a scalable manner for offering multilingual services, a necessity for cross-border complaint management within the EU. An important goal in future research is a formal approach for developing multilingual ontologies, which would allow computers to interpret and disambiguate terms in different languages through the ontology.

¹⁰ In principle, it is possible to refer to the ontology concepts by e.g. numbers, symbols, etc.

Acknowledgments. We are in debt to Peter Scoggins, Céline Damon and Andriy Lisovoy for their comments on the ideas discussed in this paper. It is also our pleasure to thank all members of TP6 for their cooperation, and particularly Bistra Vassileva, Albert Bokma, Milos Molnar, Christophe Benavent, Martin Ondrusek and Bernard Istasse. The authors are grateful to Anne Salaun, Yves Pouillet, Sophie Louveaux, Bob Schmitz, Brian Hutchinson and other topic panel members for their comments on the early draft of the CContology. We would also like to thank Prof. dr. Rita Temmerman for her comments on the methodology for the multilingual lexicalization of ontologies.

References

- [C98] Chalabi, C.: Sakhr Arabic-English Computer-Aided Translation System. AMTA 1998: 518–521
- [CIHF02] Cho, Y., Im, I., Hiltz, S., Fjermestad, J.: An Analysis of Online Customer Complaints: Implications for Web Complaint Management. In proceeding of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)–Volume 7, Hawaii, 2002.
- [CW87] Claes, F., Wernerfelt, B.: “Defensive Marketing Strategy by Customer Complaint Management: A Theoretical Analysis,” *Journal of Marketing Research*, 24, (November), 337–346. 1987
- [H01] Halpin, T.: *Information Modeling and Relational Databases*, 3rd ed, Morgan-Kaufmann.
- [G95] Gruber T.R., “Toward principles for the design of ontologies used for knowledge sharing”, *International Journal of Human-Computer Studies*, 43(5/6), (1995).
- [GG95] Guarino, N. and Giarretta, P., “Ontologies and Knowledge Bases: Towards a Terminological Clarification” in: *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, N. Mars (ed.), pp. 25–32, IOS Press, Amsterdam (1995).
- [JDM03] Jarrar M., Demy J. and Meersman R., “On Using Conceptual Data Modeling for Ontology Engineering.” In Aberer K., March S., and Spaccapietra S., (eds): *Journal on Data Semantics, Special issue on “Best papers from the ER/ODBASE/COOPIS 2002 Conferences”*, Vol. 1.1, Springer,(2003)
- [JLVM03] Jarrar, M., Lisovoy, A., Verlinden, R., Meersman, R.: “Ontoform” Ontology based CCforms demo, Delivery No: D6.8, CCFORM Project (IST-2001-34908), 5th framework.
- [JM02a] Jarrar, M., Meersman, R.: Formal Ontology Engineering in the DOGMA Approach. In: 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), *Lecture Notes in Computer Science*, Vol. 2519, Springer-Verlag, Berlin (2002)
- [JM02b] Jarrar, M., Meersman, R.: Scalability and Knowledge Reusability in Ontology Modeling, In: *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine (SSGRR2002s)* (2002)
- [S] Sowa, J.: “Concepts in the Lexicon: Introduction”, <http://users.bestweb.net/~sowa/ontology/lexicon.html>
- [VS03] Vassileva, B., Scoggins, P.: Consumer Complaint Forms: An Assessment, Evaluation and Recommendations for Complaint Categorization. In Jarrar, M., Salaun, A., (eds): *Proceeding of the “International Workshop on consumer complaint forms for online resolution machines”*, Brussels, (2003)

Some Ontological Tools to Support Legal Regulatory Compliance, with a Case Study

Aldo Gangemi¹, Alessandra Prisco², Maria-Teresa Sagri³,
Geri Steve¹, and Daniela Tiscornia³

¹ Laboratory for Applied Ontology, ISTC-CNR, Rome Italy

² Università per Stranieri, Perugia, Italy

³ ITTIG-CNR, Florence, Italy

Abstract. The increasing development of legal ontologies seems to offer satisfactory solutions to legal knowledge formalization, which in past experiences lead to a limited exploitation of legal expert systems for practical and commercial use. The paper describes some ontology-based tools that enable legal knowledge formalization. *Jurwordnet* is an extension to the legal domain of the Italian version of EuroWordNet. It is a content description model for legal information and a lexical resource for accessing multilingual and heterogeneous information sources. Its concepts are organised according to a "Core Legal Ontology" (*CLO*), based on *DOLCE+*, an extension of the *DOLCE* foundational ontology. *Jurwordnet* and *CLO* are also used to represent the assessment of legal regulatory compliance across different legal systems or between norms and cases. An example is discussed concerning compliance between EC directives and national legislations.

1 Introduction

The 80's experiences in the field of legal knowledge formalisation were mainly dedicated (especially in continental civil-law countries) to the choice of the best paradigm of representation (declarative versus deductive approach, rule-based, logic-based), while in the 90's most of the AI&Law community turned its attention to the features of legal reasoning and of the dialectic dimension of law (deontic modalities, defeasible reasoning, argument construction). Investigation on the type of entities of legal knowledge has been understated though. As a consequence, legal expert systems never came out of the level of prototypical applications, since they were lacking a solid methodology for knowledge modelling: formalising legislative knowledge was a subjective process, time- (and cost-) consuming, relatively unreliable from the user perspective, and not easily re-usable by different applications.

An ontology-based approach offers a solid support in the formalization process, as it permits the explicitation of the underling assumptions, and the formal definition of the components of legal knowledge. Accordingly, the tasks carried out in the past are being faced in a new perspective.

In this paper, of the two main streams of interest in the civil law countries: *legal advice* and *norm comparison*, we will consider here only the latter. By the way, compared to the latter, the former requires an investigation of the relation between

individual cases and common sense situations, which involves crucial problems such as value judgement, open-textured concepts, interpretation issues, etc.

The paper analyzes some ontological features of the legal domain. The use of ontology-based models is here examined in the light of norm comparison processes, with particular reference to the assessment of compliance between EU directives and national legislations, which seems to be a promising field of application. Moreover, the need of an accurate definition of the lower levels of the ontology is stressed, in order to support the conceptualization work. On this point, the JurWordnet project, a legal extension of the Italian WordNet database, will provide the lexical basis in the construction of specific domain ontologies.

2 Legal Ontologies as Description Models for Norm Comparison

The first intuitive argument for the adoption of ontologies as a description model for comparing norms is that they will provide a common (even if not neutral) language to express them, since only homogeneous entities can be compared. A further practical consideration is that most initiatives in the field of legal standard definition (LeXML, Metalex, NIR) consider legal ontologies strictly connected with the structuring of normative text. The ontology is therefore both a description model and a source of metadata for semantic tagging, providing at the same time a tool for conceptual retrieval and a model of content which maintains references to legal texts.

As illustrated in [15], norm comparison may be conceived in several ways:

As a diachronic process, norms from the same system and regulating the same domain may be compared in order to detect differences related to changes in time, or specialisations of the situations (amendment, exception, extension.)

As a synchronic process, norms of different systems, regulating the same situation, can be compared in order to assess differences in national or local policies, in regulated behaviours, in social impacts, etc.

The first process, dealing with the dynamic aspects of legal systems, requires, as pointed out in [6,15], an accurate definition of external and meta-level assumptions, defining criteria of specificity, criteria of ordering, and meta-criteria for resolving conflicting criteria. It is not completely clear, at the present state of research, how the ontology-based approach could offer new contributions to such well-known and long-time debated problem within the AI&Law Community, since a subsumption criterion seems not effective enough in detecting specialty (exceptions) when applied to the legal domain. On the other hand, a promising technique, based on theory reification and detailed axiomatization, is being tested, as briefly presented in section 4 and specially addressed by [6].

The second process assumes social relevance, as the setting up of methodologies for merging different regulations may have actual applications and produce useful results into the globalization process that involves the regulatory environment as well.

The comparison of norms regulating the same situation in different jurisdictions requires the solution of several legal issues, e.g. the completeness of the corpus (how to assess if all the norms relevant for the issue have been taken into account), the detail of granularity, the degree of legal authority of the sources (e.g. the different weight of

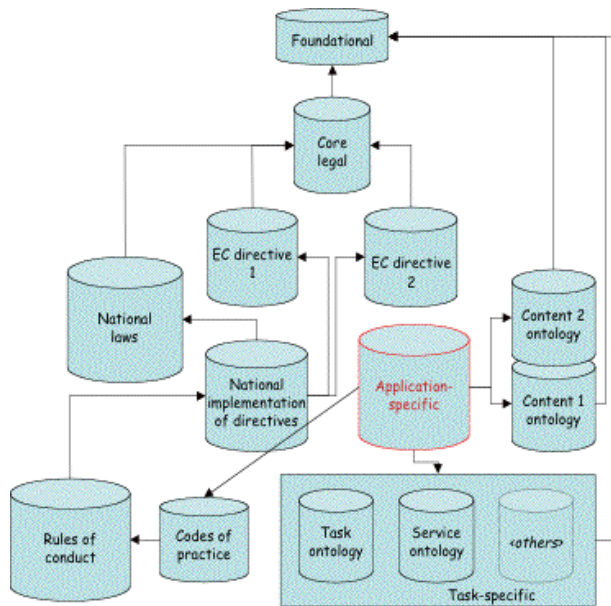


Fig. 1. An ontology library for EC Directives. Arrow semantics is theory inclusion.

case-based Law in Common Law vs. Civil Law countries). On the other hand, the relationships between European and national legislations, which we are here proposing as a test candidate, seem to be a relatively simplified field of experimentation, where several of these issues can be partly ignored.

Two possible perspectives of comparison are:

To check the correspondence between the Directive's policies and the regulative aims of the national lawmakers, underlying the national regulation that implements the Directive

To compare the national legislations adopting the same Directive in order to evaluate the level of harmonisation actually reached.

The two goals need different requirements and assumptions that we do not treat here, as our aim is to explain how a well defined description model, based on solid ontological grounds both at a lexical and conceptual levels, may be of practical utility. As a preliminary remark, the choice of European/national legislation for testing the methodology offers advantages because of:

The clear identification of normative sources to be compared (every national Act implementing an EC Directive can be clearly and autonomously identified)

The explicit assertion of the normative goals and social policies in the premises, which provides explicit criteria of analysis

The standard structure of the text, to which an already consolidated methodology for structuring and tagging legislative acts can be applied.

On the last point, we refer to our experience in the NIR project: as all national projects aimed at providing standard DTDs for structuring legislative texts, the Italian project *Norme In Rete* (NIR, [1]) has produced classes of metadata, containing

information both on the legal issues (authority, date of enactment, identifiers, references, validity) and on the textual components (typology of normative sources, hierarchical organization of the sections). The content metadata include the definition of the normative functions of norms [16] that enable to describe a text as a collection of norms, classified according to their function.

As a first step in the comparison process, text structuring "pre-processes" the normative information, in order to identify the entities involved in the regulation (definition, constitutive norms), to enable the comparison of similar classes of norms (prescription, sanctions, administrative or financial regulations), and to exclude rules dealing with the management and updating of the legal systems (amendments, cross-links), which are relevant only to a national dimension.

In comparing the normative structure of EC and national texts, it is likely to assume that most EC regulations include prescriptive rules directed to the national legislative bodies of the Member States, which should be implemented, at the national level, as prescriptions, constitutive rules, and procedures. In general, the entrenchment of norms from Directives, national laws, codes of practice, etc. can be conceptualized as represented in Fig 1:

EC Directives and *national laws* are represented in separate ontologies, which both inherit the *Core Legal Ontology* and the *Foundational Ontology* used to build the Core

The ontology of the *content domain* (social world) addressed by the directives is also based on the Foundational Ontology

The *national implementation* of directives should inherit both from EC directives and from the national laws, without being inconsistent

Rules of conduct and *codes of practice* in the Directive's domain inherit from (and should be consistent with) the national implementation of the Directive

Any compliant *application ontology* will inherit from all those ontologies, besides the basic *service* and *task* ontologies addressed by the application.

Some of the modules in the architecture will be described in the following:

The main classes of entities that populate the Core Legal Ontology

The Legal World interpretation according to the basic assumptions of the DOLCE+ foundational Ontology

The Jurwordnet lexicon, a "lightweight" ontology that approximates a large set of legal domain ontologies.

In section 5, a case of norm comparison (in a compatibility scenario) is sketched.

3 The Core Legal Ontology and Jurwordnet

The development of the Core Legal Ontology (CLO) takes into account methodologies proper of *foundational* ontologies [2][3], and proposals in the field of legal ontologies [5][25]. CLO organises juridical concepts and relations on the basis of formal (meta) properties defined in the DOLCE¹ foundational ontology [4].

¹ DOLCE stands for "Descriptive Ontology for Linguistic and Cognitive Engineering."

The basic types of entities that populate the domain of Law are assumed to be clearly identifiable and reasonably intersubjective, and, as such, they are pointed out through a minimal set of properties and relations from DOLCE and some of its recent extensions, notably the “Descriptions and Situations” ontology (D&S) [9]. DOLCE extended by D&S will be referred here as “DOLCE+”.

The methodological choices, as well as the exploitation of properties suitable for the legal domain, are based upon the approach of legal theory and philosophy of Law. For example, the most common definition of *norm* shared by legal theorists is based on the schema “fact-norm-effect”: properties (in some case the existence itself) and events inherent in the entities of the *legal world* depend on norms, whose role is that of describing generic facts (situations), and ascribing to them generic effects (legal qualifications).

As another example, in term of *speech acts* [22] norms are either *declarative* or *directive* acts: constitutive norms (declarative acts) produce a direct modification of reality, while regulative norms (directive acts) do not imply the regulated situation coming into effect. As a (legal) consequence of it, regulative norms can be violated, whereas constitutive norms can only be void.

In CLO, legal world is conceived as a *representation*, or a *description* of the reality, an ideal view of the behaviour of a social group, according to a system of rules that are commonly accepted and acknowledged. The current version of CLO is based on the DOLCE+ distinction between **descriptions** (in this domain *legal descriptions*), which encompass laws, norms, regulations, crime types, etc., and **situations** (*legal facts* or *cases* in this domain), which encompass legal states of affairs, non-legal states of affairs that are relevant to the right, and purely juridical states of affairs. This enables us to use that distinction to represent meta-juridical conceptualisations (*meta-norms*) as well.

From the logical viewpoint, DOLCE+ ontology for descriptions and situations is a first-order representation of logical theories and states of affairs [9].

In practice, a legal description (the *content* of a norm, a regulation, a decision, etc.) is assumed to be the reification $\mathcal{D}_T$ of a (potentially formalized) theory T , while a legal case $\mathcal{C}_S$ is assumed to be the reification of a state of affairs S that can satisfy T .

When we use this distinction together with the DOLCE foundational ontology, we get typical mapping functions from the elements of T into the “components” of $\mathcal{D}_T$, and from the elements of S into the “setting” of $\mathcal{C}_S$. In particular, provided that the three basic categories of DOLCE are *endurant* (including object- or substance-like entities, either physical or not), *perdurant* (including state- or process-like entities), and *region* (including dimensional spaces of attributes such as time, geographical space, color, etc.), and that DOLCE includes several primitive relations, such as *part*, *connection*, *constituency*, *inherence* of qualities in entities, *participation* of endurants in perdurants, etc., the mapping results as follows:

Perdurant entities (e.g. hearing, stabbing) in a case setting must be sequenced by some legal course of events (e.g. murder reconstruction steps, procedure to paying taxes)

Endurant entities (e.g. person, knife) in a case setting must play some legal role (e.g. citizen, witness, weapon)

Region entities (e.g. at 3pm) in a case setting must be values for some legal parameter (e.g. murder time)

Legal courses, roles, and parameters are all components of a legal description

Legal parameters are requisites for roles and courses (e.g. murder time can be a requisite for its reconstruction)

Legal roles have a modal target in a course of events (e.g. citizens are obliged to a procedure to paying taxes)

This framework is partly depicted in Fig. 2. For the sake of visual clarity, we show our ontologies in UML class diagrams, assuming a description logic-like semantics [23] for them: *classes* are interpreted as *concepts*, *generalization* is interpreted as *formal subsumption*, *associations* and *attributes* are both interpreted as *relations*.² The ontologies mentioned here are available in various languages and formats [<http://ontology.ip.rm.cnr.it>]. Figure 2 is an excerpt of DOLCE+. It is being used as a *conceptual template* in several academic and industrial projects because of its flexibility in representing the core concepts and relations in a domain.

Hence, a legal description is composed of legal roles, legal courses of events, and legal parameters on entities that result to be bound to the setting created by a legal case. This enables us to build a complex, *functional* representation of the Law and of its facts.

Since the *satisfaction* relation holding between legal descriptions and cases is the reified counterpart of the semantic *satisfiability* relation, we can specialize it in order to create a *taxonomy* of satisfiability. In fact, various kinds of semantic satisfiability can be envisaged according to the *function* a theory is supposed to describe, for example:

The way of *executing* an obligation

The way of *exercizing* a power

The way of *realizing* a desire

What is *believed* to be true

The *suggested* way to act

What is *expected* to happen

The way of *preventing* something to happen

The way of *assessing* the *conformity* of a state of affairs against a rule

The way of *assessing* the *compatibility* of two norms.

In section 4., we will show how CLO can be used to formalize two different norms, to assess their compatibility, and to assess the conformity of a state of affairs against the highest regulatory priority.

3.1 Types of Entities in the Legal World

CLO and Jurwordnet are populated by legal notions, which are represented according to the abovementioned assumptions. Here we list some of those notions and how they are represented.

² Various semantics have been proposed for UML e.g. [24]. Here we use class diagrams as a diagrammatic interface to the description logic version of DOLCE+.

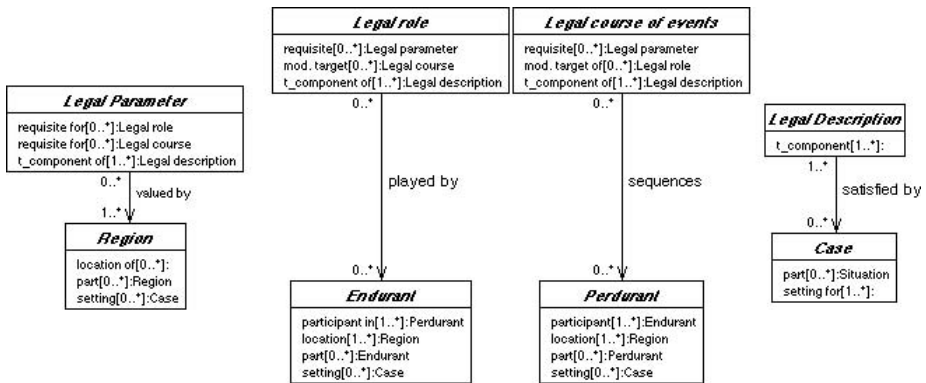


Fig. 2. A UML class diagram showing a conceptual template for legal descriptions (conceptualisations) and situations (cases)

Law, in the generic sense of the Latin *ius*, is *composed of* Norms that include social and ethical rules, practices, and conventions. Legal norm is a sub-class of norm, *expressed by* a Normative Text that is *physically represented by* a Document. Norms may even be *satisfied by* purely Juridical Situations, as for norms that describe other norms: (e.g. amendments, law making norms, validity norms.). A legal norm *functionally depends on* Legal Norms and on Collective Acceptance. Among norms, *constitutive* and *regulative* norms are distinguished; *definition* and *power-conferring* rules are sub-classes of constitutive norms. According to their type, norms may have different *parts* and *components*, for example: *Legally constructed institutions* (constitutive norms), *Legal powers* (constitutive norms), *Institutional functions* (constitutive norms), *Institutional powers* (power-conferring norms), *Behaviours* (regulative norms), *Incrimination acts* (incriminating norms).

Modal Descriptions are *proper parts* of regulative norms that contain some *modality target* relation between legal roles (legal agents involved in the norm) and legal courses of events (descriptions of actions to be executed according to the norm). The classification of Modal Descriptions is based on the Hohfeld's *Theory of basic conception* and on the *Theory of normative positions* [10]. Recent revisions of legal philosophers and logicians [11][12][13] provided a formal framework and a computational transposition of it [14]. Following Hohfeld's approach, the normative positions are mainly described throughout relations of opposition/correlation between them. In the DOLCE+ ontology, modalized descriptions reify the regulatory theories that a regulated state of affairs must satisfy. Non-reified theories are usually expressed in some deontic or action logic, as in [14], but the reified counterpart enables to talk of partial or incomplete theories, and allows reasoning on them at first-order [9]. Here we present some examples of legal modal descriptions.

Legal Right is a *social advantage* (Bentham), a *free choice* (Hart), or a *protected interest* (MacCormick); it justifies the imposition of duties, the entitlement of claims and privileges, the transfer of powers. In a wide sense, it includes subjective rights. In a strict sense, according to the Hohfeldian definition, it is correlative to *Duty* and better expressed by the term 'claim'.

Privilege is correlative to *Non-right*, *Immunity* is correlative to *Disability*. *Disability* is opposite to *Abstract Power*. *Abstract Power/Capacity* represents the generic attitude of being entitled of rights/claims, or of specific powers. In civil law systems, the *Capacity to act* is a sub-class of *Legal capacity*: in fact, a legal subject has legal capacity, but not the capacity to perform valid legal acts. *Legal power* is specifically dependent on power-conferring norms: an important sub-class is *Institutional Power*.

Faculty/Implicit Permission: in deontic logic the implicit permission derives from the absence of obligation; it differs from power because it doesn't imply the production of new legal effects. It is opposite, to non-right. *Explicit Permission* implies a liability of the permitted agent towards the agent who detects the power to permit, the adoption of the permitted goal, and the empowerment of the permitted agent in relation to the permitted action. It is specifically dependent on *Authorisation* [14].

Legal-Empowerment: it is *specifically dependent* on power-conferring norms, and it implies, as a precondition, *Potestas* (potestative right), or simple power. *Potestas* is the power to create (or modify) legal states of affairs in the sphere of other legal subjects. Its precondition is the disability of the involved subject (e.g., *patria potestas* of parents towards minor sons), or the willing of the *involved* subject, as *expressed* by an act of delegation and *represented* by a mandate. The *opposite* of *Potestas* is *Liability* (not of the directed bearer of obligation, but of the involved subject).

Legal Roles are descriptions of functions endorsed by either physical or non-physical objects. Among legal roles, some of them constitute the basic entities of the legal world, such as *legal subjects* and *legal assets*. Legal-subject is an *agentive* legal role, while legal asset is *non-agentive*. Physical existence is a sufficient but not a necessary condition for being a *legal subject*: legal agent is therefore a *role*, created by (constitutive) rules and played either by human or social individuals. A *natural legal person* is a legal subject just because of its physical existence (even before birth and after death), whereas the *legal person* role needs to fit strict requirements, such as age, mental non-illness, or artificial existence. Legal person implies legal subject, but not the contrary. Agentive legal roles are distinguished into 1) Legally Constructed Institutions, specifically dependent on constitutive norms, perform legal acts, and are created by constitutive norms that justify their existence and validity, e.g. Ministry, Body, Society, Agency, and 2) Legal Functions, which are played by legal subjects.

Legal Information Objects *depend on* agents' cognitive states and *represent* legal descriptions. For example, Expression of Willingness may be not only a Linguistic Object (an Oral Expression), but this can also be manifested with behaviours. Legal Documents are legal information objects that depend on some Physical Representation (paper, electronic form, etc.); there are cases in which a certain form is a requirement for the valid existence of a Legal Act.

Legal Cognitive Objects are internal descriptions, (e.g. *agreement* and *mistake*), which are *results of* mental processes or which *embody* cognitive states. Cognitive objects are specifically dependent on agentive physical objects (e.g. a natural person).

Among **cognitive states** (that are *perdurants*), intentionality is subsumed by will, which is subsumed by consciousness. For example, *Suitas* (Free will), based on the distinction between will and intentionality, grounds the distinction between fault and intentional fraud in crime law.

Legal Facts (including *cases*) are situations *depending on* norms (only facts relevant to legal systems are legal facts). Some subclasses are *Natural facts* (e.g. death) vs. *Human facts*, that depend on consciousness (but not on will), which can be distinguished into: 1) *Institutional facts*, satisfying constitutive rules, 2) *Legal acts* (in a strict sense), depending on will, 3) *Legal transactions*, depending on intentionality, 4) *Crimes*, which satisfy incriminating norms.

3.2 A Lexical Extension: Jurwordnet

Jur-(Ital) Wordnet (Jurwordnet, [7]) is a formal ontology-based extension to the legal domain of the Italian version [8] of *EuroWordNet*. Its motivation comes from the NIR project, to which Jurwordnet provides a metadata lexicon for semantic tagging. Jurwordnet can also be used as a support tool for information retrieval systems, in order to facilitate the access to heterogeneous and multilingual data³, and a conceptual source for information extraction, automatic tagging, etc.⁴

Jurwordnet leverages on WordNet, Eurowordnet, and DOLCE+, envisaging a multilingual legal ontology, at least as far as the lexical knowledge of Law is concerned. With reference to norm comparison, Jurwordnet can be considered a link between the domain ontologies and the legislative texts, since it provides a wide coverage of legal concepts and their lexical realizations.

Jurwordnet is still under development: we expect to reach a satisfying coverage of the basic legal contents through the definition of about 3000 synsets. The enrichment of the lexical database will probably act as a testbed for the ontological level.

4 Compatibility between Entrenched Norms: An Example

According to [18], «a significant part of legal reasoning can be considered as contradiction handling». Until recently, formal methods applied to contradiction handling have been limited to the exploitation of nonmonotonic reasoning, belief change, and similar techniques. Hierarchical structure of legal systems has also been indicated as a possible source of conflict resolution [18]. Hierarchical structure is based on:

1. Source ordering (*lex superior* prevails), based on authoritative entrenchment
2. Specialty ordering (*lex specialis* prevails), subordinated to source ordering

³ The Project LOIS (Legal Ontologies for Knowledge Sharing), aims at creating a Jurwordnet in five European languages.

⁴ The Jur-WN methodology will be tested in the E-Psinet Project (e.Content Program), aimed at comparing the regulatory environment of Public Sector Information in the EC.

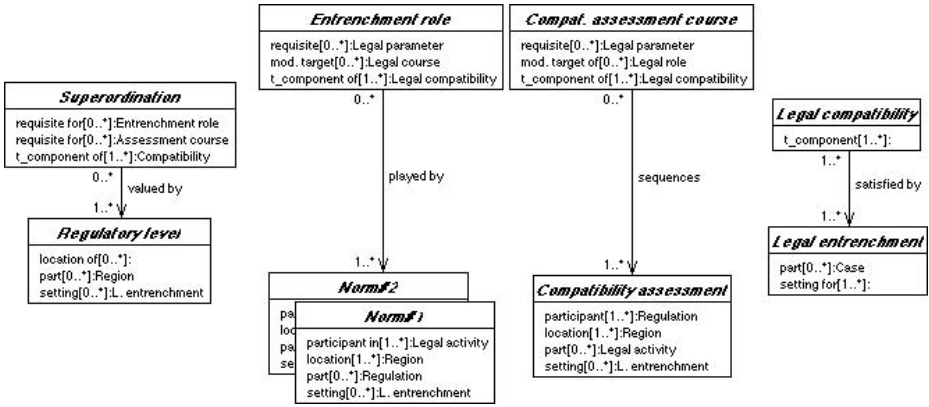


Fig. 3. A D&S conceptual template for the legal compatibility scenario

3. Chronological ordering (*lex posterior* prevails), subordinated to specialty ordering
4. Domain ordering, where a legal domain can be superordinated to a subdomain

Contradiction detection between norms can be considered as a special case of *norm comparison* [6][15], and it does not necessarily lead to *norm incompatibility*, since there exists an entrenchment of norms [19] that derives from the hierarchical structure of legal systems and that preserves compatibility. Hierarchical compatibility requires a *local* consistency of the logical theories representing legal norms.

In the *theory reification* framework of DOLCE+ and CLO, which was partly present in [6], norms are first-order entities, then their possible logical *inconsistency* disappears, and migrates into class *disjointness*. Compatibility is then assessed as a case of conformity between a *compatibility scenario* and a situation including a set of norms. By *conformity* we mean that a case satisfies a legal description. In case of ground regulative norms (norms not involving other norms in their satisfying cases), like banking regulations analyzed in [6], a social state of affairs must *conform* to a norm in its legally relevant setting. In case of more abstract norms, conformity is assessed against states of affairs that can contain other norms. Compatibility assessment is one of those cases: two or more possibly incompatible norms should conform to a compatibility meta-norm (or principle, ideal, scenario, etc.).

In Fig. 3, a *compatibility scenario* is represented as a specialization of the CLO conceptual template. Legal compatibility is represented as a kind of legal description that can be satisfied by a legal entrenchment situation whose setting includes certain regulatory levels for pairs of norms, according to *superordination* parameters, *entrenchment* roles, *compatibility assessment* courses, etc. Hierarchical structuring is represented as superordination parameters valued by regulatory levels according to the source, specialty, time, and domain of norms. Norms play some entrenchment role according to their hierarchical position. The algorithm to assess compatibility is specified in a compatibility assessment course.

Each norm involved in legal compatibility can be exploded into a CLO template that represents its satisfiability conditions, until ground norms have been reached.

Despite the hierarchical structuring of norms, legal compatibility is not always satisfiable in legal theory (*norm dynamics*), as well as in jurisprudential practice, leading also to the problem of *alternative interpretations*. The compatibility scenario can be used to assess or simply to represent those cases. In the following, a relatively simple example from theoretical jurisprudence is presented.

4.1 Non-mediated Applicability of an EC Directive

European Court of Justice has repeatedly stated the superordination of European norms against national ones. In order for this superordination to be enacted, a European norm needs an explicit acceptance from a national government.

On the other hand, due to the delays from the national governments, Luxembourg judges have continuously stated the validity of the *direct application* of European norms, provided that the European directive is *clear, unconstrained, and beyond terms*. The first statement of this principle is present in the Van Duyn judgment [17], which states that: «where the Community authorities have, by directive, imposed on Member States the obligation to pursue a particular course of conduct, the useful effect of such an act would be weakened if individuals were prevented from relying on it before their national courts and if the latter were prevented from taking it into consideration as an element of Community law».

More recently, this principle has been extended to all administrative bodies of Member States, which have the duty of applying a directive's provision, even by disregarding *non-conforming national norms*. Obviously, these principles hold only *vertically*, since a Member State cannot transfer the consequences of its delayed adoption to an individual citizen.

Member States have mostly recognized those principles, for example the Italian Constitutional Court has definitively recognized the superordination of EC norms to Italian ones, especially after the judgment 64/90 [20], stating that whenever the directives are sufficiently precise and unconstrained, and their adoption is still missing beyond the terms of enactment, nevertheless they can be used to contrast any national norm that does not conform to them.

Figure 1 shows a modular architecture that covers a reasonably complete ontology library for EC directives. Modularizing ontologies is a first step towards the formal representation of entrenched norms, in order to maintain *local* consistency. But in order to assess the kind of relation between entrenched norms, or even to assess their compatibility, a *strongly modular approach* is needed. By “strongly modular” we mean a contextual awareness at the level of norm content, which is obtained by exploiting the D&S framework in DOLCE+ [9]. We will show the use of strong modularity to represent the Enasarco judgment [21] of the Italian *Corte di Cassazione*, the highest Italian judicial Court.

In the Enasarco judgment, the *Corte di Cassazione* states that the EC Directive 653/86, which excludes the obligation for commercial agents to be enrolled in a dedicated registry, is *directly applicable* to Italian jurisprudential system. The problem arose from a decree obtained by Enasarco (the institution that manages the pension system for commercial agents) against a company that did not pay any pension tax to some commercial agents. The company objected that, since those

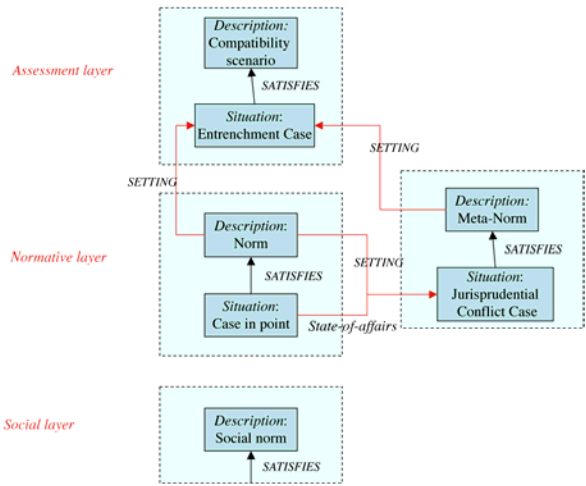


Fig. 4. Epistemological layering in compatibility jurisprudence

agents weren't enrolled in the registry, then, according to the Italian law 204/85, art. 9, stating that agents not enrolled in the registry are not allowed to act like commercial agents, the company is not obliged to pay those taxes. The preliminary judicial Court rejected the decree on the basis of the company's objection. But the *Corte di Cassazione* agreed with the counter-objection of Enasarco on the basis of the EC Directive 653/86.

In this judgment, the *Corte di Cassazione* has applied the superordination of *source* and of *chronology* to the entrenched norms, even in absence of a *specialty* superordination that was lacking from Italian legislative bodies. Consequently, the compatibility scenario is preserved according to:

1. The EC 653/86 and the Italian 204/85 laws, potentially in conflict within an integrated (and non-reified) logical theory
2. The meta-norms stated by EC Court of Justice, for example in the Van Duyn judgment, as well as in its interpretation dedicated to the 653/86 (stated in February 18th, 1988), and stating that national norms cannot subordinate the validity of agency contracts to the agent's enrolment in a dedicated registry
3. The superordination parameters derived from the hierarchical structure of norms presented in [18]
4. The resolution procedures adopted by Courts
5. The attributes of the parties involved in the judgment.

Saying that the compatibility scenario is preserved amounts to finding that the legal entrenchment between the potentially conflicting norms actually *satisfies* a particular legal compatibility description. This case also shows a peculiar feature of the DOLCE+, namely its capability of expressing an *epistemological layering* (EL). EL appears whenever a theory is satisfiable by a model that includes elements that are models or theories on their turn. This is a notoriously hard logical problem, and requires turnarounds to be implemented in an efficient way.

The D&S (and CLO) solution is to represent EL as a scenario in which a description is satisfied by a situation that contains in its setting some other situation or description, thus constraining EL reasoning at first-order. In the Enasarco judgment, EL (Fig. 4) derives from the representation of meta-norms (interpretations or advises from previous judgments, usual superordination of norms, etc.), which are satisfiable by cases in which norms are in the setting (this is also the case of the compatibility scenario, which belongs to a further *assessment* layer). Norms included in that setting are on their turn satisfiable by cases in which social objects (or even social norms) are included, and so on, until the EL mechanism runs out of the scope of the Legal World.

4.2 Conclusions

We have introduced some projects, resources, and methods that are being developed in a joint collaboration between the Laboratory for Applied Ontology and ITTIG-CNR. The general methodology applied in the collaboration uses formal ontology techniques and resources to formalize legal knowledge and the legal lexicon. Applications based on the same techniques are being built to perform comparison of diachronically distinct norms in the same domain [6], to detect compliance of synchronically distinct norms, to control the conformity of activities against previous agreements or contracts, and to support Semantic Web searches. In the paper we have introduced: the DOLCE+ foundational ontology, on which a Core Legal Ontology is being defined; the Jurwordnet lexicon based on CLO, and an example of how to use CLO to represent judicial Acts delivered in presence of incompatible norms.

Compatibility is shown as distinct from logical inconsistency, and we explain how it can be mapped to a first-order classification of concepts. Nevertheless, the ontological representation of compatibility does not ensure *per se* either an effective obtainment by individuals, or the actual exploitation of legal compatibility in a widespread legal system. We only claim that exploitation and effectiveness would be aided in principle by our formal methods. In our example, Enasarco got the right to be paid by the company, but the Italian Court has no power to impose to Enasarco to actually *use* those pension taxes to provide incomes to the future retired agents. In other words, formal assessment of legal compliance and conformity would result in a weapon in the hands of large organizations, unless this technology is coupled with e-government policies aiming at integrating legal compliance with social enactment.

References

- [1] Report on “*Il progetto Norme in rete*”, Rome, www.normeinrete.it/documenti, 2000.
- [2] Gangemi A., Guarino N., Masolo C., Oltramari, A., Schneider L. 2002. Sweetening Ontologies with DOLCE. in *Proceedings of EKAW 2002*, Sigüenza, Spain, pp. 166–178.
- [3] Gangemi A, Pisanelli DM, Steve G, “An Overview of the ONIONS Project”, *Data and Knowledge Engineering*, 1999, 31, 1999.
- [4] Masolo C., Borgo S., Gangemi A, Guarino N, Oltramari A, Schneider L, “The WonderWeb Library of Foundational Ontologies”, <http://wonderweb.semanticweb.org>.

- [5] Visser P., T. Bench Capon, Ontologies in the Design of Legal Knowledge Systems, towards a Library of Legal Domain Ontologies, in *Proceedings of Jurix 99*, Leuven, Belgique, 1999.
- [6] Gangemi A., Pisanelli DM., Steve G., A formal Ontology Framework to represent Norm Dynamics. Proc. of *Second International Workshop on Legal Ontologies*, Amsterdam, 2001.
- [7] Sagri M.T., Progetto per lo sviluppo di una rete lessicale giuridica on line attraverso la specializzazione di ItalWordnet, in *Informatica e Diritto*, ESI, Napoli, 2003.
- [8] Roventini A., Alonge A., Bertagna F., Calzolari N., Girardi C., Magnini B., Marinelli R., Speranza M., Zampolli A. (in press), *ItalWordNet: Building a Large Semantic Database for the Automatic Treatment of Italian*, in “Linguistica Computazionale”, Pisa-Roma, ISSN.
- [9] Gangemi A., Mika P. 2003, *Understanding the Semantic Web through Descriptions and Situation*, Meersman R, et al. (eds.), *Proceedings of ODBASE03*, Springer, Berlin, 2003.
- [10] Kanger S., *Law and Logic, Theoria*, 38, 105–132, 1972.
- [11] Lindhal L., *Position and Change. A study in Law and Logic*, Reidel, 1977.
- [12] Pörn, I., *Action Theory and Social Science, Some Formal Models*, Reidel, 1977.
- [13] Jones A. and Sergot M., *A formal characterisation of institutional power*, *Journal of IGPL*, 4(3), 429–445, 1996.
- [14] Jones A, *A logical Framework*, in J.Pitt (ed.), *Open Agents Societies: Normative Specifications in Multi-Agent Systems*, Wiley and S., 2003
- [15] Boer A., van Engers T. and Winkels R., Using Ontologies for Comparing and Harmonizing Legislation, in *Proceedings of the 9th ICAIL Conference*, Edinburgh, 2003.
- [16] Biagioli C., “An XML editor for Legislative drafting”, *JURIX Workshop on E-Government*, 2002.
- [17] European Court of Justice, Van Duyn vs. Home Office (41/74), judgment Dec. 4th, 1974.
- [18] Sartor G., “Legal Reasoning and Normative Conflicts”, in *Reasoning with Inconsistency*, 1991.
- [19] Gärdenfors P., “The Dynamics of Normative Systems”, 1989.
- [20] Italian Constitutional Court, judgment 64/90, 1990.
- [21] Italian *Corte di Cassazione, Sezione Lavoro*, judgment #3914, March 18th, 2002.
- [22] Austin J., “How to Do Things with Words”, Harvard University Press, 1962.
- [23] Baader F, et al., “The description logic handbook”, Cambridge University Press, 2003.
- [24] Guizzardi, G, Herre, H, Wagner G, “Towards Ontological Foundations for UML Conceptual Models”, *ODBASE’02*, Irvine. Springer, Berlin, 2002.
- [25] Breuker J, and Winkels R, “Use and reuse of legal ontologies in knowledge engineering and information management”, *ICAIL03 Wks on Legal Ontologies and Web-based Information Management*, Edinburgh, <http://lri.jur.uva.nl/~winkels/legontICAIL2003.html>, 2003.

Regulatory Ontologies: An Intellectual Property Rights Approach

Jaime Delgado, Isabel Gallego, Silvia Llorente, and Roberto García

Universitat Pompeu Fabra (UPF), Departament de Tecnologia,
Pg. Circumval·lació 8, E-08003 Barcelona, Spain
{jaime.delgado, isabel.gallego, silvia.llorente, roberto.garcia}
@upf.edu

Abstract. Regulatory ontologies combine ideas and results from different fields. Our approach to the topic is based on two aspects. First, the work we have done in the development of an ontology for Intellectual Property Rights (IPR), based on previous work done by our group and others. The second aspect is to apply these results to a new regulated area, that of the Data Privacy Protection. The paper describes the IPR ontology (IPRonto) and analyses how we can proceed in the area of Data Privacy Protection.¹

1 Introduction

Digital Rights Management (DRM), or the wide issue of protection and management of Intellectual Property Rights (IPR), is a key aspect for the real and full deployment of multimedia content through the net. We refer to multimedia content that has associated property rights.

Although several solutions exist, most of them are closed solutions and do not interoperate with other approaches. To solve this problem, many efforts are being done to standardise the way to approach DRM, without limiting the freedom of application developers, but facilitating interoperability. It is worth mentioning here the new standards being specified by ISO/IEC through its MPEG group (ISO/IEC JTC1 SC29/WG11) [1], namely a Rights Expression Language (REL) [2] and a Rights Data Dictionary (RDD) [3], that will be parts of the MPEG-21 standard.

Those standards, or similar initiatives in other organisations (see for example [4] for more information), should facilitate the implementation of applications allowing protection, distribution and control of copyrighted content. However, for sophisticated applications that might need an extra intelligence to interpret rights (for example, when negotiating rights between two parties), a more abstract representation of IPR information is needed. For this reason, we have developed an ontology to help us in the representation of IPR information, thus easing complex implementations. This ontology, IPRonto [5], is described in detail in section 2, the major part of the paper. There are other initiatives in the development, learning and modelling of ontologies, such as [6].

¹ This work has been partly supported by the Spanish Ministry of Science and Technology (TIC2000-0317-P4-05 and TIC2002-01336)

On the other hand, there are other aspects also relevant here, apart from the technical ones just mentioned. In particular, IPR is something that is normally regulated, something that applications do not always take into account. In order to help solving this problem, we could express these regulations in a formal way. A clear candidate approach for this expression is the use of ontologies, giving place to regulatory ontologies. Although IPRonto does not follow a specific law, it tries to cover the entire IPR domain. It would be easy to produce versions of the ontology following specific laws.

Both from a technical and a legal point of view we came to the objective of developing regulatory ontologies. Now, we could move this approach to another domain. The domain we have selected is that of Data Privacy Protection. Section 3 of the paper introduces how we will deal with the issue, again both from a technical and a legal point of view.

2 IPRonto: An Ontology for Intellectual Property Rights

In order to improve the management of Intellectual Property Rights (IPR), there is a need for a common language for IPR representation in the open and global framework provided by the Web. This language is aimed to help building a reliable Web where intellectual property rights can be managed in an open, global and adaptable form, so people can share, sell, buy, etc. multimedia content subject to IPR, depending on their needs. A semantic approach seems a more flexible and efficient way of achieving these activities than a syntactic one. We have developed an ontology for IPR, IPRonto [5], that puts into practice this approach.

Current initiatives focus on a syntactic approach, the formalisation of some XML DTDs and Schemas that define rights expression languages. The semantics of these languages, the meaning of the expressions, is formalised separately into term-definition dictionaries where definitions are given in natural language, solely for human consumption and not easily automated.

Our idea is to facilitate the automation and interoperability of IPR frameworks by integrating both parts, called Rights Expression Language (REL) [2] and Rights Data Dictionary (RDD) [3].

This can be accomplished using ontologies. They can provide the required definitions of the rights expression language terms in a machine-readable form. Thus, from the automatic processing point of view, a more complete vision of the application domain is available and more sophisticated processes can be carried out.

Moreover, the modularity of web ontologies allows its free extension and adaptation without losing the connection to previous roots.

We have two views for our IPRonto ontology, the Static one and the Dynamic one. Section 2.1 explains the different entities that constitute the static view of IPRonto, while section 2.2 describes the events, processes and subontologies related to the dynamic view.

2.1 Static View of the IPR Ontology

We can see the static view of IPROnto as a tree where elements are related from the bottom to the top. In the next subsection we introduce a skeleton of the tree where only key elements, for simplicity, are presented. In the rest of subsections, some of the key elements are detailed to give a more complete view of the ontology.

IPROnto Key Elements of the Skeleton

As already said, this skeleton ontology, shown in Fig. 1, only contains some of the key elements. The root of the tree is an Entity, which may be Physical or Abstract. In turn, a Physical entity may be an Object or a Process, being this one more interesting, which might be an Event or a Situation.

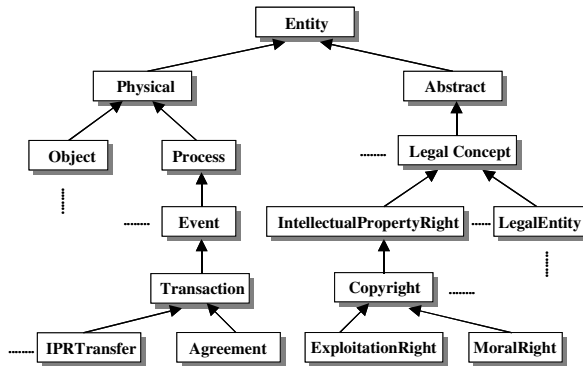


Fig. 1. IPROnto key elements of the ontology skeleton

In the other side of the tree, although several elements may belong to an Abstract entity, only the Legal Concept is presented in the skeleton. Nevertheless, other options, not sketched here, are possible, such as Relation or Quantity. While the Legal Concept might come from a few elements, only details are given for two cases: Intellectual Property Rights and Legal Entity. The presence of dotted lines in the tree means that other “brother” elements exist, but have not been included in the skeleton.

The most interesting part of the IPROnto related with the problem of the privacy of the rights, is the part related to the Legal Concept Entity, specifically Intellectual Property Rights Entity and their derived rights. These rights are associated to multimedia digital content for automatic commerce in the Internet, in special the Author’s Rights or Copyright. All elements of the tree below Legal Concept, together with all leaves, are detailed in the following subsections.

IPROnto Legal Concept

Fig. 2 shows the Legal Concept together with its dependent elements.

A Legal Concept is a concept defined by law, statute or international convention. Its terms are generally understood and defined in a series of international conventions and treaties and under national laws.

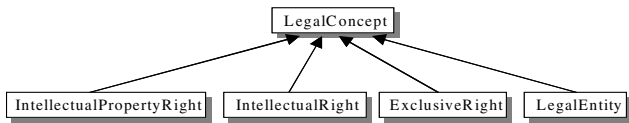


Fig. 2. IPRonto Legal Concept

LegalEntity

An entity possessing capacity in law to exercise or enjoy an intellectual property right is a LegalEntity. It can be divided into two categories, Corporate Legal Entity, which refers to a legal entity inside a corporation and a Natural Legal Entity, which could also be seen as a person in the real world.

The starting point, from an IPR’s point of view, is the selection of the model in which to base IPR representation and negotiation. The IMPRIMATUR Business Model [7], the one we selected, identifies a series of entities that may take different roles, such as Creator, Provider, Rights Holder, Distributor, IPR Data Base, or Watermarking & Fingerprint marking (see Fig. 3).

- 1. Creator: Owner of the copyright in a work. He/She holds the exclusive right to use or authorize others to use the work on agreed terms.
- 2. Rights Holder: Provides a license to exploit the creation on terms which may be either predefined subject to negotiation.
- 3. Content Provider: It acts in name of the Rights Holder, compiles and packs creations to provide to the distributors with multimedia content.
- 4. Media Distributors: Apply to a Rights Holder for a license to exploit the creation.
- 5. Customer: Person who wants to make use of a creation.

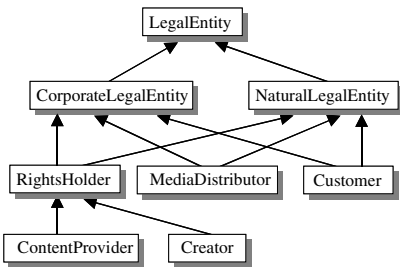


Fig. 3. IPRonto Legal Entity

IPRonto Intellectual Property Right

Intellectual Property Rights (IPR) are legal instruments that provide a limited monopolistic right to the owner of things such as patents, trademarks or copyrighted works. They provide an incentive for the creation of and investment in new works (music, films, print media, software, performances, broadcasts, etc.) and their exploitation, thereby contributing to improved competitiveness, employment and innovation.

Some of the Intellectual Property Rights considered in IPRonto are shown in Fig. 4. This description is based on the Berne Convention [8] and the WIPO Copyright Treaty [9]. It is, as the sources state, broadened enough to ensure

international applicability. There are local initiatives to implement these recommendations, as the EC Directive on Copyright 2001/29/EC [10] or the US Digital Millennium Copyright Act [11].

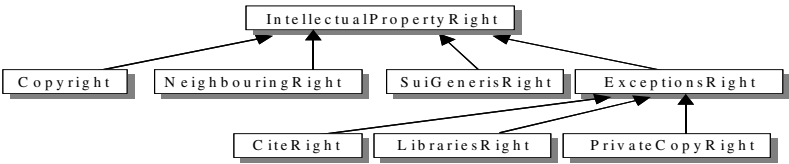


Fig. 4. IPRonto Intellectual Property Right

In the context of copyrighted works, IPR includes author’s rights, which will be referred to later, sui generis rights (applicable for databases) and neighbouring rights (especially concerning artist-interpreters).

IPRonto Copyright

All these rights are automatically given to originators of works (creators) by the simple fact of their authorship. They include Moral rights that are independent of the author’s economic rights and even after their transfer and Exploitation rights (economic rights), which are oriented to guarantee financial profit to originators of works.

Figure 5 shows the relationship between Copyright, ExploitationRight and MoralRight.

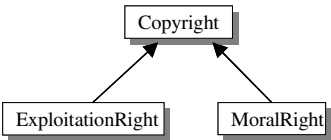


Fig. 5. IPRonto Copyright

Moral Right

Moral rights are independent of the author’s economic rights, and even after their transfer, the author has the following rights:

- 1. Dissemination Right: Exclusive right to disclose the work.
- 2. Paternity Right: Exclusive right to claim authorship of the work.

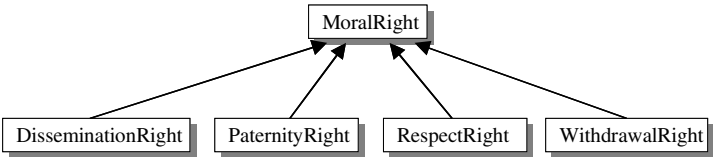


Fig. 6. IPRonto Moral Right

3. Respect Right: Exclusive right to object to any modification to the work prejudicial to his reputation.
 4. Withdrawal Right: Exclusive right to withdraw the work.
- Fig.6 shows the MoralRight together with its dependent rights.

Exploitation Right

The complete ExploitationRight subtree is shown in Fig. 7.

Exploitation rights (so called economic rights) are oriented to guarantee financial profit to originators of works. They include:

1. Reproduction Right: Exclusive right to authorize the reproduction, direct and indirect, permanent or temporary, in any manner or form.
2. Communication to the Public Right: Exclusive right for the authorisation of any communication to the public of their works. These includes that members of the public may access them from a place and at a time individually chosen by them. Examples are: public performance, broadcasting, interactive on-demand transmission, etc.

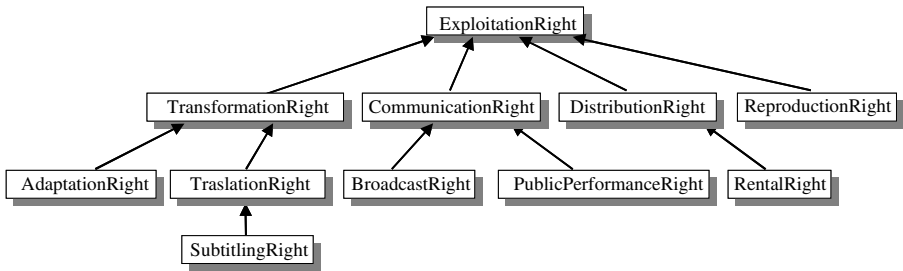


Fig. 7. IPROnto Exploitation Right

3. Distribution Right: Exclusive right to authorize the making available to the public of the original or copies of the work by sale or other transfer of ownership. Relevant only to tangible objects.
4. Transformation Right: Exclusive right to authorize the manipulation of their works in any manner or form.

2.2 Dynamic View of the IPR Ontology

The dynamic view of IPROnto allows the construction of IPR business models for its use in electronic commerce. In the following sections, some of the events that can occur in the IPR domain are described. Moreover, we describe the creation process with a subontology derived from IPROnto.

Events

The concepts and relations defined in IPROnto can be used to represent the typical processes of the IPR domain, and also the agents and resources they involve. These representations can be automatically tested to check semantic consistency, thanks to

the explicit semantics that emerge from the ontology structure of concepts and relations.

To show the expressive power of this semantic IPR approach, we present a detailed view of the creation life cycle. Fig. 8 contains some kind of flow diagram relating the different actors involved in the creation life cycle and the transitions that transfer exploitation of this creation between them. This generic business model has been inspired in the one defined by the IMPRIMATUR Project [7].

In the next clauses, we give a detailed view of the events that provoke each of these transitions. The events are represented using the concepts and relations defined in IPROnto and other reused ones. This is done by means of a graph of nodes, i.e. concepts, and edges, i.e. relations.

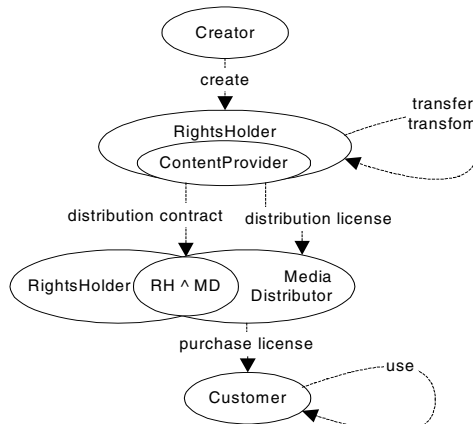


Fig. 8. Content life cycle

Finally, it is also worth noting that the whole content life cycle of a particular creation can be retrieved from the concatenation of the diverse events that provoke the transitions in its life cycle, and thus from the representations we made using IPROnto.

Create Event

The Create Event represents the intellectual property creation, starting the life cycle of this intellectual property and its associated rights. The new creation may be totally new or based on previous ones, so the life cycle of the new creation will affect the used source creations, e.g. some royalty fees might be provided to the source creations rights holders.

Fig. 9 shows the central concept, the “Create” event that relates all the involved concepts, some of which are:

- Agents: the creator, playing the role originator, and a rights holder that promotes the creative event.
- Result: a creation resulting from the creative process.
- Context: a time and place where and when the event takes place.

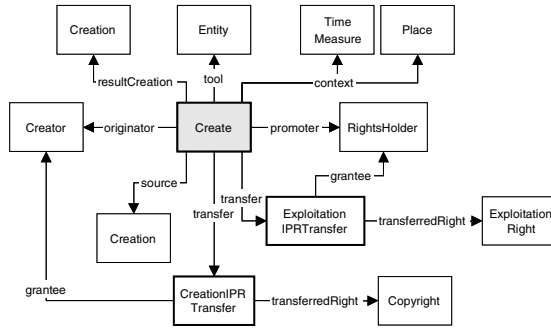


Fig. 9. Create Event

- **Transfer:** relates the main event to the associated events that model the actual transfer of rights provoked by the create event. In this case, all the Copyrights are given to the creator and a subset of them, not including Moral Rights, to the promoter. Therefore, the creator and the promoter rights holder share the Exploitation Rights on the new creation. In both cases there is no granter of rights because they emerge as a result of the creative process.
- **Input:** the source creations, if they exist, on which the resulting creation is based.

IPRAgreement Event

The “IPRAgreement” event represents a generic contract for rights transference between legal parties that hold rights, the “RightsHolders” (Fig. 10):

- **Agents:** two persons consent the agreement, they may be the same individuals as the involved “RightsHolders” or their representatives.
- **Transfer:** a connection to the rights transfer process that models the actual transference of rights agreed in the referencing “IPRAgreement”, this is an associated event.
- **Context:** there are two contexts, one for the agreement and the other for the rights transference. The latter may detail a period of time and a geographical region when and where this transfer would be valid.
- **Creation:** the creation over which the transferred rights apply.
- **Compensation:** the counterpart that the rights granter receives for the granted rights.

Distribution Contract Event

The Distribution Contract is also a kind of “IPRAgreement”. It is established between a “ContentProvider”, or more generally a “RightsHolder” and a “MediaDistributor”. As some rights are transferred, the recipient media distributor becomes also a rights holder. Therefore, the concept playing the role “grantee” of the “DistributionIPRTransfer” must be a media distributor but also a rights holder, thus we use the intersection “ $\wedge$ ” symbol to represent this. Finally, in this type of contract the transferred rights are the Distribution, Reproduction and Communication Rights. The other involved concepts have more or less been explained in the previous events. See Fig. 11 for a graphical view.

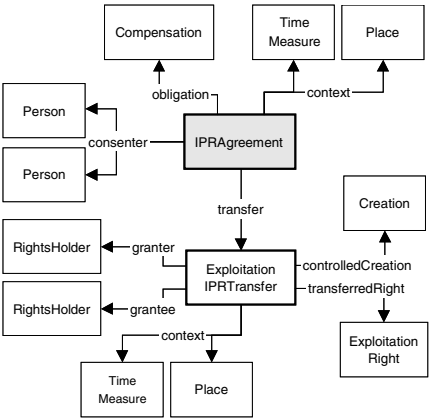


Fig. 10. IPRAgreement Event

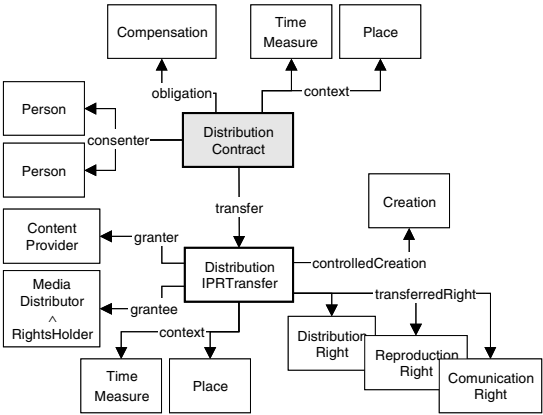


Fig. 11. Distribution Contract Event

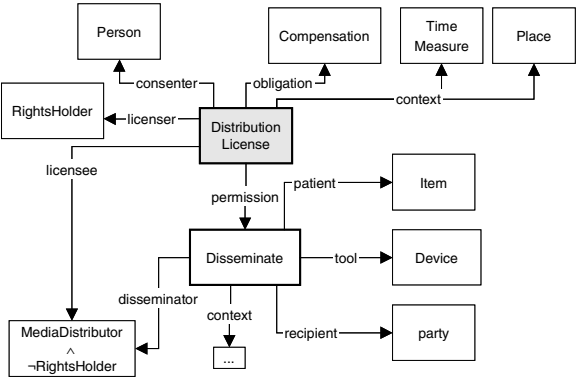


Fig. 12. Distribution License Event

Distribution License Event

Fig. 12 shows the Distribution License Event.

We talk about licenses, nor contracts or the more generic “IPRAgreement”, when the agreement between two parties does not involve a transfer of rights, but only a licensing of certain actions on the creation. The “DistributionLicense” is a kind of license in which a rights holder authorises the dissemination of a certain creation in a determined set of conditions to a “MediaDistributor”. As there is not transfer of rights, the licensed party must not become a rights holder.

Purchase License Event

This is the kind of license that appears at the end of the creation life cycle. It is established between a final user, the customer and the distributor. The license authorises a determined use under certain conditions. See Fig. 13 for a graphical view.

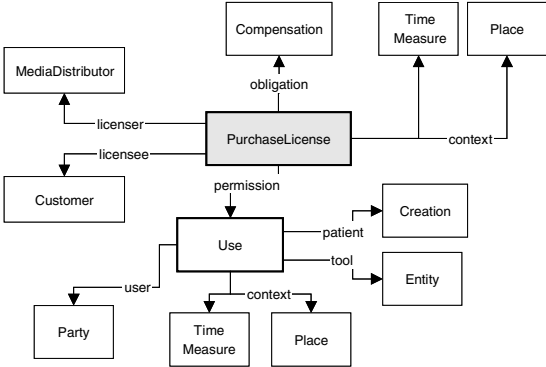


Fig. 13. Purchase License Event

Subontologies

The hierarchy of concepts and relations from the static view of the ontology, plus some of their interrelation constraints, has to be translated to a machine aware form. This leads to situations in which the relationship between concepts is difficult to explain in a two-dimensional graph. For this case, we have developed the concept of subontology, that goes one step further from the static view.

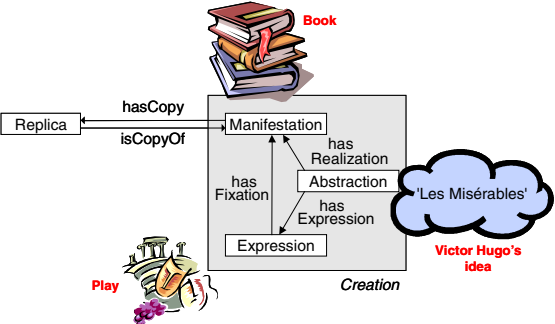


Fig. 14. Creation Subontology

We can see in Fig. 14 a clear example, the Creation subontology (the union of its abstract, material and temporal forms). These are the three points of view of a creation, the abstract, object and event perspectives that relate it to the upper level ontology from SUMO [12]. For instance, if we take the creation “Les Misérables”, we observe its object view in a script, a book, etc. At the same time, its film projection would be the event part and all (script, book, film...) have in common an abstraction that comes from the original Victor Hugo’s idea.

3 Data Protection and Privacy

The concepts presented throughout this paper not only can be applied to IPR but also can be considered in the field of data protection and privacy.

In this area we find two completely different viewpoints: the legal viewpoint and the technical viewpoint. In the next sections, it is explained in more detail what do we exactly mean when we separate data protection and privacy issues into these two viewpoints.

3.1 Legal Viewpoint

From the legal point of view, several things can be outlined in this area. First of all, there is a completely different approach in the regulation of data protection and privacy between European Community (EC) and United States (US).

While US defends the self-regulation between companies, EC developed the European Data Protection Directive 95/46/EC [13] for regulating the protection of personal data and the corresponding transposition of this directive (for example, for Spain [14]). One of the articles of this directive obliged to the states inside the European Community to transpose the directive into national laws, but this has not already fully happened, as some countries do not have this national law yet [15]. This can give us an idea of the complexity of working with data protection and privacy legislation even for the experts in law definition (governments), since the directive was already approved in 1995.

There is also another directive related to personal data protection, the European Directive on privacy and electronic communications 2002/58/EC [16]. This directive applies to the personal data that can be obtained and used in the telecommunications sector. This directive has also to be transposed into national laws by each of the members of the European Community by the end of 2003.

Nevertheless, due to the pressure done by EC for the protection of their citizens’ data, the US Department of Commerce defined safe harbour privacy principles [17] for guaranteeing data protection of European citizens’ private data as described on the 95/46/EC directive.

3.2 Technical Viewpoint

From the technical point of view, privacy and data protection is a key issue as many organisations are working in the definition of mechanisms that help the provision and control of data privacy to end users, specially for data being accessed through the Web. Protection of personal data in off-line environments is a different problem, but by no means less difficult. However, in this paper we concentrate in the Web situation.

In this area, the first initiative we have to mention is the Platform for Privacy Preferences (P3P) project [18] from W3C. In the specification for version 1.0 of P3P [19] the syntax and semantics of P3P privacy policies is defined, and also the mechanisms for associating policies with Web resources. They describe a mechanism for defining new data elements and data sets and contemplate the possibility of adding extensions. Also inside the P3P project, we find APPEL (A P3P Preference Exchange Language) [20], for expressing preferences over P3P policies. These preferences are defined in APPEL as a set of rules. A user agent can make automated or semi-automated decisions regarding the acceptability of machine-readable privacy policies from P3P enabled Web sites based on APPEL rules.

However, P3P is only one initiative for privacy policies definition. Many web sites have their own privacy policy described in legal clauses on *plain text*. Users of that site should read these clauses before providing their private data, as they can be considered as a contract between the user and the web site. It is hard to read all legal clauses inside the privacy policy of a web site and this is why P3P is being defined. Nevertheless, P3P has several drawbacks. For instance, it does not allow the access to individuals to the personal data stored in the web sites offering P3P versions of their privacy policies.

Also inside the technical viewpoint, we should mention the CEN/ISSS Workshop on Data Protection and Privacy (CEN/ISSS WS/DPP) [21]. This workshop, just kicked-off in July 2003, follows the conclusions of *The Initiative for Privacy Standardization in Europe (IPSE)* report [22], the aim of which was to analyse the current status of privacy protection efforts and determine whether standardisation actions in the broadest sense could benefit the processes and implementation of the European Data Protection Directive 95/46/EC [13]. The report, prepared by an IPSE Project Team and approved by the IPSE Steering Group, looked at the background and a broad representative sample of the current work in progress on data protection.

Finally, a description of possible use of Digital Rights Management systems to meet the challenges of Privacy Rights Management can be found in [23].

3.3 Alignment of Legal and Technical Viewpoints

Our approach for aligning data protection and privacy from the legal and technical viewpoints is the definition of a common vocabulary for describing the regulations present in this field in the European Union [13,16].

Our objective is the definition of a simple ontology of data privacy terms. To do so, we will base our work on the experience in ontology definition obtained during the development of IPRonto [5], the ontology about IPR described in section 2.

For not starting from scratch in the definition of the simple privacy ontology, we are waiting for the results of the CEN/ISSS WS/DPP [24], mentioned before. One of

the objectives of this workshop is the definition of a generic set of contract clauses reflecting the requirements of Articles 17 of Directive 95/46/EC [13]. The results of this work might be very useful for defining our ontology, possibly contributing it to the Workshop. Afterwards, we could also align the simple privacy ontology with P3P, bearing in mind that there are many tools for automatic checking of P3P policies and APPEAL rules.

Also based on our previous work done in the construction of a framework for negotiation of IPR [25] and on the idea of data licenses for the use of personal data using the P3P language described in [26], we will describe a negotiation scenario for the use of personal data by web sites. It is possible to use our negotiation framework as we have defined it for being able to use different data structures.

4 Conclusions

We have presented an ontology for IPR that is being used for developing applications that involve interchange of multimedia content having associated rights. Since IPR is a regulated domain, the approach we have followed to develop the ontology could be used to adapt it to the different regulations, thus simplifying the implementation of systems conformant to laws.

We are currently developing applications that use IPROnto, for example, in the context of IPR negotiation. In our research group, Distributed Multimedia Applications Group (DMAG) [27], we have done some example implementations in the NewMARS and AgentWeb projects [28, 29].

Furthermore, we want to go one step forward and extend again the approach to a new domain, that of Data Privacy Protection (DPP). The high level of activity in this field, and the strong need of having systems that follow DPP laws, seem to confirm that there is a need for this. We have introduced in the paper how we are going to deal with this problem.

Finally, it is also worth mentioning new plans we have in the area. The international standards under development on DRM (see Section 1) specify a set of DRM terms and their relationships and possible extensions, providing a kind of regulation. We are working in a new ontology, that will probably be a formal subset of IPROnto, that would formalise the REL and RDD standards from MPEG-21.

References

1. Moving Picture Experts Group (MPEG) ISO/IEC/ JTC1 SC29/WG11 <http://www.chiariglione.org/mpeg/index.htm>
2. ISO/IEC FDIS 21000-5 “MPEG-21 Rights Expression Language (REL)”. ISO/IEC JTC 1/SC 29/WG 11/N5839, July 2003
3. ISO/IEC FDIS 21000-6 “MPEG-21 Rights Data Dictionary (RDD)”. ISO/IEC JTC 1/SC 29/WG 11/N5842, July 2003
4. Delgado, J., Gallego, I. and Rodríguez, E. “Use of the MPEG-21 Rights Expression Language for Music Distribution”, WEDELMUSIC 2003, IEEE Computer Society
5. IPROnto, <http://dmag.upf.es/ontologies/ipronto>
6. ONTOWEB project, <http://www.ontoweb.org>

7. IMPRIMATUR Project, <http://www.imprimatur.net>
8. Berne Convention for the Protection of Literary and Artistic Works, <http://www.wipo.int/treaties/ip/berne/index.html>
9. WIPO (World Intellectual Property Organisation), <http://www.wipo.org>
10. Directive 2001/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonisation of certain aspects of copyright and related rights in the information society, http://europa.eu.int/eur-lex/en/consleg/pdf/2001/en_2001L0029_do_001.pdf
11. US Digital Millennium Copyright Act, <http://www.loc.gov/copyright/legislation/hr2281.pdf>
12. SUMO (Suggested Upper Merged Ontology), <http://ontology.teknowledge.com>
13. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection on individuals with regard to the processing of personal data [...], http://europa.eu.int/comm/internal_market/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf
14. Spanish Law, LEY ORGÁNICA 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal, Boletín Oficial del Estado número 298, 14th December, pages 43088–43099
15. Analysis and impact study on the implementation of Directive EC 95/46 in Member States, http://europa.eu.int/comm/internal_market/privacy/docs/lawreport/consultation/technical-annex_en.pdf
16. Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector, http://europa.eu.int/eur-lex/pri/en/oj/dat/2002/l_201/l_20120020731en00370047.pdf
17. U.S. Department of Commerce: Safe harbor privacy principles, <http://www.export.gov/safeharbor/SHPRINCIPLESFINAL.htm>
18. Platform on Privacy Preferences (P3P) Project home page, www.w3c.org/P3P/
19. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, <http://www.w3c.org/TR/P3P/>
20. A P3P Preference Exchange Language 1.0 (APPEL1.0), <http://www.w3c.org/TR/P3P-preferences/>
21. CEN/ISSS Data Protection and Privacy, <http://www.cenorm.be/iss/Workshop/DPP/default.htm>
22. The Initiative for Privacy Standardization in Europe (IPSE), http://www.cenorm.be/iss/Projects/DataProtection/IPSE/ipse_finalreport.pdf
23. Kenny, S. and Korba, L., “Applying digital rights management systems to privacy rights management”, *Computers & Security* Volume 21, Number 7, November 2002, pag. 648–664
24. Business Plan for CEN/ISSS Workshop Data Protection and Privacy (CEN/ISSS WS/DPP), <http://www.cenorm.be/iss/Workshop/DPP/BPv2.pdf>
25. J. Delgado, I. Gallego, R. Garcia and R. Gil, “An Architecture for Negotiation with Mobile Agents”, *MATA’02*, Springer Verlag, LNCS, vol. 2521, pp. 21–31, 2002.
26. S. Cha, Y. Jung, “From P3P to Data Licenses”, 3rd Workshop on Privacy Enhancing Technologies (PET2003), Springer Verlag, LNCS, vol. 2760, 2003.
27. DMAG (Distributed Multimedia Applications Group), <http://dmag.upf.es>
28. NewMARS project, <http://hayek.upf.es/newmars/index.html>
29. AgentWeb project, <http://dmag.upf.es>

Copyright Conflicts Resolution: A Method Based on Legal Ontologies in an Architecture Enabling Subsidiarity

Melanie Dulong de Rosnay^{1,2}

¹ Law University Paris-II Panthéon
CERSA - Research Centre for Administrative Science
10 rue Thénard, 75005 Paris, France
Computer Science, Law and Linguistic Laboratory
<http://www.cersa.org>

² MEDIALIVE
20 rue Saint Didier, 75116 Paris, France
melanie.dulong-de-rosnay@medialive.fr
<http://www.medialive.fr>

Abstract. Legal and technical regulations both serve to allow a wide dissemination of creative works through networks. Technological measures and rights management information implement copyright 'contracts' and are based on Rights Expression Languages, i.e. ontologies that link copyright concepts and terms to events occurring in digital networks.

While implementing such devices, specific attention shall be dedicated to the compatibility between different stakeholders' expectations. In order for Digital Rights Management systems to be fair, they must adequately support opposing interests; notwithstanding and in all cases the law requires that copyright exceptions granted by national legislations be respected.

After a description of the legal framework and current technical standardization initiatives, we will detail a method for obtaining ontologies from syntactical parsing of copyright law legal texts. This ontology will be supported by a client-server architecture capable of resolving conflicting rights expressions including the use of the principle of subsidiarity.

Keywords: Legal ontologies, Rights Expression Languages (REL), copyright law, exceptions to exclusive rights, Digital Rights Management (DRM) systems, technological measures and rights management information, multimedia standardization, MPEG-21.

1 Introduction: Users Expectations

Innovation in information technologies, digital compression and telecommunications facilitates public access to creative works. Intellectual property rights management, especially copyright, is becoming an automated regulator of new technologies. Authors, original creators of works distributed on networks, have the exclusive right to authorize the reproduction and the communication of their works, within certain

limits. They share this prerogative with content producers, according to licensing terms sharing the benefits from the commercial exploitation of their works. The combination of compression technologies, network capabilities and digital rights management offers a great opportunity for a wide, quick and low cost dissemination of creative and information works.

Authors benefit from these innovations at several levels. They will find a large audience for the dissemination of their works in an environment where several barriers are removed:

- Audiences are no longer restricted to national markets and linguistic areas;
- Authors can benefit from distribution platforms, providing a large visibility, metadatas for an enriched access to content according to end-user's preferences;
- Authors can also publish and distribute directly their works through personal websites or community portals.

They expect Intellectual Property Management and Protection (IPMP) systems to express and enforce the rights they decided to attach to their works. Their expectations can be financial but also related to the moral control of their works' destination, some initiatives being "devoted to expanding the use of creative work available for others to build upon and share" [34]. Identification, marking and watermarking systems ensure a persistent protection of their moral rights, at first attribution rights i.e. paternity recognition. Authors also expect to enjoy an easy access to works of other authors' as a source of inspiration, or as material to be reused and integrated to their own works, according to new creation ways (sampling, VJ, cut-up techniques, tributes, covers...). IT can provide interfaces facilitating contacts between creative communities; and DRM are expected, if not to support, at least not to restrict those exchanges.

End-users expect a user-friendly and easy access to culture, education and entertainment while preserving their privacy, as producers and distributors are looking for a return on investment, reliable event reporting, and cost-effective management. Producer's representatives are afraid of peer-to-peer file sharing consequences on their sales. They express publicly their disapproval by asking for Internet service technical providers and Courts support. Public institutions, open source communities and academics fear that DRM systems, acting as electronic fences, replace legal public mediation and restrict access to works.

2 Technical Standardization

Electronic commerce is faced to hurdles [3] that prevent the full development of digital media and services. Digital media creation, sharing and distribution has to be supported by intelligent DRM systems, able to adapt to new use models and to cope with contradictory interests of all stakeholders. DRM solutions should therefore be able to process dialog, arbitration, mediation between those expectations expressions. It should also allow easy update of content rights management policy, according to expectations and time-to-market strategies evolution.

From Consumer Electronics manufacturers and industry point of view, bits shall be delivered at low cost, technical standardization of delivery formats enabling interoperability between systems and devices. The Moving Picture Expert Group [19] develops standards for coded representation of digital audio and video allowing the

development of mp3, digital television, DVD, information retrieval and description ... Launched in June 2000, MPEG-21 [20] plans to define a global framework to enable transparent and augmented use of multimedia resources through the delivery chain, from content creator to end-user, across networks, equipments and communities. Their answer consists in an IPMP architecture structuring identification, management and protection of multimedia works. The integration of those modules is not only faced to technical requirements, but also to emerging uses and business models. A machine-readable language able to express rights, which are getting more and more complex in the digital world, defines algorithms enforcing electronic content governance within a trusted environment. Code is Law, and software developers have to be aware of the value they embed inside their code and architectural design [9], [10]. Computed access control may stand for legal code. Current DRM systems are based on interoperable languages describing terms and conditions for digital resources manipulation, transport and usage. Such languages shall be able to express simple and complex rights expressions, based on standardized Rights Data Dictionaries.

3 Rights Expression Languages

Grounded on Imprimatur European project results [27], <indecs> and <indecs>2rdd [26] provide a methodology for the interoperability of metadatas supporting electronic commerce. Rights and permissions basic structures are expressed by a Context Model defining the relationships between entities, organized around the notion of Action on a Resource.

ODRL (Open Digital Rights Language [28] supported by IPRSystems [29] developed another semantic on the standard XML. Rights governing Assets are including Permissions characterized by Constraints, Requirements and Conditions. Rights are expressed by Parties through Offers, Agreements and Revoking Rights.

XrML (extensible Rights Markup Language [30]) is developed at Content Guard [31]. XrML source code is proprietary and can be licensed to DRM applications. An Authorization Context is organized around fundamental notions: a Right can be performed by a Principal on a Resource under specific Conditions.

IPROnto [32] formalized at DMAG [33] on a semantic web approach, structures IPR information at a higher level of abstraction. This ontological domain representation expresses IPR roles and events capturing the various steps of the content lifecycle: creation, rights transfer, distribution, use, reproduction, transformation, providing a broad framework for Automatic Negotiation. Two consensers are involved, the Requester and the Licensing Agent, representing the rights holders. Offers and Counteroffers expressed with IPROnto help to reach an Agreement for a defined time period, compensation and action [4].

Several other initiatives developing open source or proprietary RDD and REL shall be pointed out: OASIS [22], Open Mobile Alliance[23], IEEE Learning Technology Standards Committee [24], Internet Streaming Media Alliance [25]...

4 Legal Requirements on IPMP Architectures

Our position elaborated in context of MPEG-21 requirements [21] claims that only a minimum core of compulsory features, the lowest common denominator of worldwide copyright legislation and contractual practices, should be specified within a standard that aims at defining an interoperable framework for future DRM applications.

We will first present legal requirements with regard for the hierarchical relationship between international conventions and national legislations. Legal norm provides their own exceptions: prior rights holder's authorization is requested before use, but unauthorized uses of a copyrighted work do not always constitute an infringement. We will then describe the consequences of those legal requirements on IPMP architecture, and propose several architectural features:

1. The exclusive right of authorizing can be materialized within MPEG-21 through an 'Event Request' made by the user/client to the user/rights holder.
2. True negotiation between users is made possible through the implementation of a Rights Expression 'bi-directional dialog', formalizing contractual conditions.

4.1 National Copyright Expressions

National copyright expressions allocate different rights to entities, and entities are defined differently. For instance, the qualification of a work legal category (audiovisual or multimedia work, database or software, are handled differently) has implications on work's legal status and similar works will receive different treatments and induce different remunerations according to the state they are linked to. These local differences have consequences on rights representation. Negotiation conditions will vary from one case to another and jurisdiction is not always simple to determine in case of international sales and law conflicts. Reasoning automated systems (eg. authorization, royalties sharing, fair use scope, compulsory licences) shall be compatible and adapted at all levels as embedded concepts may have different interpretation, deserving ad hoc handling.

4.2 International Legal Principles

Some principles or pieces of legislation are internationally recognized. At the top of copyright norms international hierarchy, we find WIPO Berne Convention for the Protection of Literary and Artistic Works enacted in 1886 [13]. This text is enforceable in most countries and has been transposed into most national copyright legislations. Therefore, it is considered as a minimum common legal foundation which principles are embedded within national copyright laws and copyright management systems. Berne Convention [13] states "Authors of literary and artistic works protected by this Convention shall have the exclusive right of authorizing the reproduction of these works, in any manner or form". According to section 106 of US Copyright law [15], copyright owners have the exclusive right to do and to authorize works reproduction, alteration, distribution... This 'exclusive rights' internationally shared concept means that rights holders are, if no other provision is applying, the sole actors allowed to fulfil or authorize the accomplishment of specific acts related to

their works. Even if it is not always desirable [11], prior authorization from the rights holder shall normally be requested before any analogue or digital action, such as digital display, reproduction, communication to the public, performance, distribution, rental, translation, adaptation... Authorization is then granted for a defined use context in exchange of a remuneration.

Nevertheless, this general principle confronted to reality presents several adaptations. International and national legislations present a lot of exceptions that avoid the user to have to ask the rights holders' prior authorization every time before s/he wishes to perform an action on a protected content. This prior authorization legal paradigm might be reversed one day: average use would be free and exclusive rights would become the exception for commercial use. But it can pragmatically be assumed that content industry and rights holders will not easily abandon this prerogative.

Limitations on exclusive rights have various explanations, from technical reasons to fair use and limit the obligation to ask for authorization before an action. Temporary acts of reproduction, which are part of a network transmission technological process, are exempted from prior authorization (Europe), but not in all countries (USA). Favourable translation licensing conditions can be granted to developing countries [13]. In order to avoid having to ask prior authorization before any act, because it is burdensome or materially impossible to ask every time (radio broadcasting...), or because it is compulsory to proceed through collective management in certain areas (cable distribution in Europe). Those mechanisms are called compulsory licences and equitable remuneration is redistributed through collective societies.

A natural limit to copyright is its expiration: the work belongs to the public domain after an evolving amount of decades. Authors can also restrict their exclusive rights on a voluntary basis and place their creations into the public domain, or voluntarily restrict their exclusive rights to commercial uses [34]. Fair uses (critical or academic use...) are generally admitted if they do not affect normal exploitation or rights holders' legitimate interests, but it is up to national states to precise them according to stakeholders' requests.

If an exception to exclusive rights applies, an act that would deserve prior authorization is not considered as an infringement and rights holders shall on no account restrict content access or usage through DRM. But an exception can be enforced, meaning an event will not require a prior authorization, only once the accurate sphere of use has been defined and qualified, and not before. So, it looks impossible to formally represent a priori fair use cases.

4.3 Negotiation Architecture

Notwithstanding limitations on exclusive rights prerogatives, we can assume that all DRM systems and associated REL will translate prior authorization international requirement for commercial distribution. After an authorization request, rights holders will refuse or allow the requested act under specific conditions. The agreement is materialized through a temporary licence sent back from the rights holder to the client. Of course, there are many existing and valid models that work differently from this general way, and architecture requirements shall, rather than limit choices, try to satisfy the highest requirements level. If proposed licensing terms do not fulfill user

expectations, s/he should be allowed to express her/his own particular conditions and confront them online to the rights holder disputed licensing terms. Nevertheless, including the expression a priori within the resource does not match exclusive rights limitations qualification rules. Exceptions to the general case are identified within a use context, not according to a content or a category or user and exceptions scope cannot be persistently attached to the digital resources: unlike to use conditions, they cannot be determined by advance according to content type or rights holder's wishes. It is on the use case context that the application of an exception is recognized and prior authorization avoided.

Classical client-server architectures permit to open a dialog between users. Not only rights holders shall be allowed to express and propose use conditions, leaving end-users with no other choices than approving all the licensing terms or refusing them all and ending the transaction. The end-user shall be made able to propose alternative Rights Expressions and confront them to rights holder's Rights Expressions. As DRM shall cope with both general and specific use cases in a scalable approach, all rules will not be applied in most cases and shall not burdensome the system.

Sometimes, decisions shall be taken at a high level whereas in other cases, they can or even need to be taken at a lower level. Subsidiarity is a concept that might help to determine the ad hoc level where a policy will receive the best management and therefore to distribute competences between decisional architecture levels, deduced from an efficiency assessment. The principle of subsidiarity was first expressed in XIIIth century by Thomas Aquinas and recalled by constitutional federalists (Germany, Swiss, Italy, European Community...). Larger entities will not intervene if a lower level is able to reach the same end. High-level entities action is legitimated only if their added value enhances the inadequate capacity of lesser entities. They are supposed to withdraw from the regulation process as soon as the lesser level is able to continue on its own again. Subsidiarity can be vertical in relationship with governance levels. It can also have horizontal applications as criterion to appreciating the ability to decide and rule of social groups, for instance private and public regulation. Subsidiarity as common sense rule may guide tasks, competences and activity scope sharing between actors (law, standard, market, individuals...) or elements of any abstract architecture. As far as standards and DRM architectural design are concerned, this concept might also help to choose the right level of management and decision.

An architecture with many levels would fairly translate and support this approach. A first central server would send the resource and its associated Rights Expression to many other decentralized servers, which would adapt these Rights Expression to local requirements. Such a second level server can be an application, a distribution platform or a trans/national network. It could be dedicated to private documents sharing within a family and offer the broadest scope of Rights Expressions: all actions are allowed, and no authorization or reporting mechanisms between registered members may burdensome access to content. On the contrary, if the second level server is a commercial application, it would request a fee for a determined category of event, all other events being declared forbidden. Providing a whole range of licenses would allow to handle every situation with the lowest level of control. Any user and right holder would choose the application corresponding to its needs. Each level would define its own rules and desired level of control on actions according to legal, business, social norms and uses. At first, we find a first Rights Expression requesting

a prior authorization, to be applied if nothing else is specified. Local servers adapt the first Rights Expression to, for instance, national laws. Prior authorization condition is skipped if the Rights Expressions from the top meets a Rights Expressions from a lower level that exempts the specific action from prior authorization (public domain, compulsory licence, fair use...).

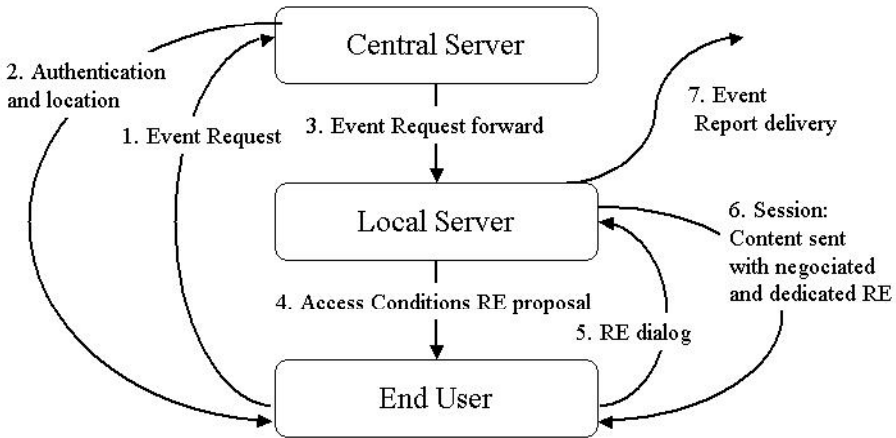


Fig. 1. Proposed IPMP architecture enabling Rights Expressions (RE) dialog.

Having gathered possible terms and concepts for Rights Expressions, a knowledge database confronting top rights holders Rights Expressions with lower end-user Rights Expressions is being elaborated and will be implemented in a secured technology (see part 5.4). According to intended use, and in order to allow a dialog between levels, Rights Expressions have to be formally represented in an ontology able to solve Rights Expressions conflicts through confrontation, negotiation, decision and solution enforcement (licence agreement, copyright exception, nothing) between Rights Expressions set up by the rights holder and use conditions expressed by the lower local server. The benefit of including a dialog allows real world negotiation conditions between users. Communication is allowed in both directions (i.e. with a return path) and not only reduced to an offer from the rights holder to the end-user, who would have no other choice than block acceptance or refusal. If the Rights Expressions issued by the right holder do not grant printing permission, the end-user might well communicate his request to the server.

5 A Legal Ontology

Even if efforts have been made by 1996 international treaties [17], [18] and 2001 European Directive [14], copyright legislations are not harmonized. Their interpretation is subject to changes and uncertainties, which are unfavorable to investments and commerce development. Legal concepts are open texture concepts [8] as their understanding is related to external conditions such as the context. This

diversity generates difficulties for the implementation of DRM systems. Indeed, it is important to express the conditions attached to a work's use with accuracy and flexibility.

5.1 Objectives

Ontologies can be defined as referential abstractions providing notions and concepts of a domain with a formalization of the links between those concepts [7]. This modeling is useful for a computer-aided exploitation of the knowledge representation. Our research project is based on the development of a legal ontology that will allow expressing rights and using situations attached to a context and a work. As it is possible to build several ontologies for a single legal domain, the composition of our ontology will be deduced from the task it is designed for [1], i.e. copyright conflict resolution and copyright balance maintenance within creative content electronic delivery.

The preliminary phase of our work was to study previous results of other Rights Expression Languages (RELs) and dictionaries. Most relevant initiatives are XrML, ODRL, <indecs> and IPRonto. Except the latter one, all projects started from use cases and not from legal texts in order to define a core set of semantics.

The added value of our methodology is to start from a legal expertise of the field, defined as the intersection between copyright law and law for IT and media. This expertise allows overtaking a partial interpretation of the texts, knowledge of legal doctrine and case law enabling to throw light on texts interpretation. Our reasoning process aims at modeling all use cases and re-use cases, the latter aspect being outside other projects' scope. We also included public service concerns to commercial and business requirements. Archival or re-use of protected content in a library, a museum or for teaching material are definitively part of content electronic delivery targets and shall be taken into account [6], [12]. Other REL give priority to commercial uses and hardly tackle fair use because making a work available for specific uses may cause security wholes for unauthorized persons. REL are agnostic and are deemed to be able to express all uses and legislations, at least indirectly or by default. However, a statistical study of the vocabulary captured by existing REL terminologies allowed to detect an under-representation of the terms and concepts attached to end-users and copyright exceptions, in comparison with expressions describing rights holders, contractual conditions and remuneration sharing.

Our contribution proposes to restore copyright initial balance by proposing a scope of terms that should be large and representative enough to describe and express core situations and specificities. The model associates to a delivery technology a metadata system enabling a legal processing of the access requests according to authorization contexts. After implementation, it will be possible to apply relevant national legislation in order to allow or turn down a request and to automate rights holder proportional remuneration. The metadata system, based on a copyright ontology, could permit to non-lawyer users to describe their status and works requested use conditions. The term 'user' is understood as all groups of actors of the value chain, from authors and creators to end-users, including broadcasters and intermediate aggregators, as defined in MPEG-21 specifications [21].

5.2 Method

We will present the method used to choose and organize concepts. An ontology dedicated to rights description and rights conflict resolution shall be able to describe at a meta level and undoubtedly national copyright legislation and case law subtleties. Classes shall be intuitive enough for a self-categorization by non-lawyer users. Links between classes (users, rights, conditions, constraints...) are representing digital content life-cycle steps from creation to licensing, fair use and reuse.

The concepts of the domain ontology are selected from legal texts currently in force. They are extracted after a statistical processing enabled by syntactical analysis software Syntex developed at Syntax and Semantic Research Team, Université Toulouse le Mirail, France and already tested on a legal corpus [2]. The analysis has been carried out on different jurisdiction levels: international (8 WIPO conventions and treaties), regional, i.e. European Community (9 European Directives), national, restricted for the moment to France Intellectual Property Code, Artistic and Literary Property part, legislations from other countries will be integrated at a subsequent parsing.

The software Syntex provides a statistical and a syntactical analysis on texts in French and in English in order to identify terms. Terms can be viewed by lists of verbs, nouns, adjectives, adverbs, noun phrases, verb phrases, adjective phrases and adverb phrases associated to their frequency and location in the parsed text.

Knowing that concepts are identified by terms or groups of terms, it appears that extracted noun phrases are more meaningful than verbs or adverbs, which will be used in a subsequent phase of the work in order to qualify the links between concepts. Syntactic and coordination relations between extracted terms will help to identify and validate semantic relations between legal concepts.

We started to work on legal texts for an obvious reason: those texts are referred to for contractualization and litigation, which are the two legal phases happening just before or right after the moment we try to represent, i.e. copyright conflict resolution within creative content electronic delivery. This method has the advantage of working on a corpus that is already organized with a logical structure dividing law articles. Working manually on textbooks table of contents would have produced limited results compared to the computed-aided method, supplying statistical information about the relative importance of terms. The comparison of their frequency in texts from different jurisdictional levels enlightens on the position that a concept or a class of concepts may occupy within the ontology and the architecture, reflecting the balance between analogue and electronic processing levels. The frequency also helps to confirm choices of terms made through previous domain knowledge.

5.3 Results

The results provide up to 2000 terms by query, depending on the text length and on the requested syntagm. All results are not used, empty terms are manually eliminated while redundant and synonym terms are gathered. The statistical study of noun phrases helps to locate, mark and link the terms that shall be part of the ontology. The domain ontology aims at gathering exhaustively terms used to describe use situations

and their legal consequences in terms of legal processing automation, which can be summarized in answering the following questions.

What is the action use case? Is this action authorized? Is an authorization request required? Who are the relevant rights holders? What are the remuneration sharing criteria? Those questions were deduced from the attitude to be adopted when wanting to use or re-use content based on professional legal practice. They are to be answered before knowing how to handle a protected work.

It appeared that French code top frequency terms are connected with royalties distribution and sharing out by collective societies. We deduce that this task should occupy a high-level position in the ontology classes and delivery architecture. The second family of terms, gathered through field knowledge, enabling the constitution of classes of terms, as first level of structuring of the ontology, are actors categories. Audiovisual and neighboring rights holders show a higher frequency. We can assume that those categories of right holders, if not being more important than the other, deserve a specific attention. More sentences being dedicated to them means that they are not to be treated like other actors. Then, the slots associated to those categories of right holders will be different from the average slots common to all actors (i.e. name, location, contact). A knowledge database editor (Protégé-2000, Stanford Medical Informatics at Stanford University School of Medicine) helps to organize the concepts into classes and to associate instances (i.e. rights, conditions, constraints) to them. Here is an excerpt presenting terms categorization. Terms regrouping enabled to obtain legal classes and their associated instances.

Around 100 concepts have been formulated after automatic processing and terms merging. A lot of work is remaining before obtaining a complete ontology. Coordination relations and verbs analysis will allow to structure classes. Further study of contractual relationships will enable to determinate the links between the classes and therefore to refine the architecture.

USER	WORK	RIGHT	CONDITION	CONSTRAINT	COMPLAINT
Author	Phonogram	Communi-	Fee	Location	Copyright
Producer	Videogram	cate	Quantity	Format	infringement
Collective	Music score	Copy	Levy	Watermark	Access refusal
society	Database	Distribute	Time	Authentication	Paternity
End-user		Translate	Registration	Berne three-step	infringement
Teacher		Aggregate	Open source	test	Non- portability
Librarian		Criticize	Attribution		
Journalist		Back-up			

Fig. 2. Excerpt of terms classification prior to ontological representation

5.4 Implementation in a Secured Technology

The described model proposes to the user the ability to confront alternative Rights Expressions to the rights holder’s Rights Expressions.

The ontology will be associated to Medialiving secured technology. Medialive technical protection consists in a separation of the original video flow into two separate parts: the first part is named contains 99% of the original video data while the other 1% is replaced by lures which make it impossible to be understood by

human eyesight and hearing. The second part contains the missing 1% of original data and is stored in a secured server. Unlike to current watermarking and encryption technologies, Medialiving avoids disseminating all the content on line, by removing pseudo-randomly small pieces of the digital stream and substituting them by lures that are compliant to compression standards (MPEG-2, MPEG-4, JPEG, AVC...). The lured digital stream can be freely disseminated and displayed but cannot be enjoyed by humans. Extracted pieces are distributed in real-time when playing the content through an automatic access control server allowing original digital stream reconstitution.

Medialiving technology prototypes ensure a secured access control after identification and authentication of the person or entity that wants to access the content. Rights holders have the advantages of a trusted system protecting the transaction. They should not hesitate any longer to entrust valuable content to the networks. This lack of confidence from content holders has been identified as a brake to e-commerce, producers being hesitant and even reluctant to digitize their assets, fearing to lose the control and the opportunity to make money [5].

The coming association of a secured technology and an ontology including fair uses is expected provide a balance between the different stakeholders. Illegitimate users might be disheartened while fair uses will be granted in a user-friendly environment. The ontology prevents the access control system from abusing end-users by preventing uses that would have been unregulated in the analogue world. DRM systems are often criticized because they put too much control on actions and events that should remain freely authorized [9].

The client-server architecture embodies real life negotiation conditions, a contract being the result of a consensus between informed parties. Parties from each side will be able to express their expectations and will. Works will be shared according to the intention of all parties and in respect of the different interests and legislations involved. Private copy is deemed to be possible because the lured file is made available to the public without restrictions. As this file cannot be enjoyed without missing data sent after identification, it can be stated that it is not a full copy because any user who want to access the content has to be identified by the server. On the other hand, the server is able to authenticate a user and to recognize a registered user. Private copy is therefore allowed in the sense that only the user that has been authenticated once will be authorized to receive missing data as many times as accorded by legal or contractual conditions.

Conflict resolution between parties Rights Expressions is theoretically possible. Providing an upstream information channel or a return path within digital content delivery architectures might help to reverse the trend of consumers preferring peer-to-peer networks (user-friendly but of irregular quality) to official platforms, suspected to favor producers to the detriment of end-users, by making private copy and interoperability between rendering devices impossible, except by circumventing technical protection, which is constitutive of a copyright infringement under WCT [17], WPPT [18], DMCA [15] and 2001 European Directive [14].

6 Future Work

The presented work should be put in context of other rights management systems as W3C Semantic Web. A larger sample of legal texts shall be parsed, and legal corpus shall be processed through other linguistic tools in order to compare the extracted terms and deduced results and complete the ontology. Besides, the terms shall be translated back into French and other languages in order to validate the porting into various legal systems. The coherence of the ontology is largely submitted to its validity in both Common Law and *droit d'auteur* systems. Comparative law principles, associated to legal translation and terminology methods, will allow correcting and refining the results.

The development of a prototype by Medialive engineers, integrating the ontology in a knowledge database deciding to deliver missing data or not, will make it possible to carry out tests on various use cases representing legal classes. Such scenarios should be elaborated in close connection with stakeholders and DRM users. They will combine business, professional, educative and private use cases and rights built upon the ontology, such as Video On Demand, embedding of a previous work in a composite work, private copy, communication without the possibility of copying... Algorithms will be developed to implement the conflict resolution process.

Domain conceptualization within an ontology can also help users to appreciate a DRM and evaluate its completeness and embedded values, control, trust or privacy level. This aspect can be helpful when a user has to choose between different DRM.

Acknowledgments. We would like to acknowledge CERSA and MEDIALIVE for supervising and funding our research within MPEG-21 stimulating framework.

References

1. Trevor Bench-Capon, Task Neutral Ontologies, Common Sense Ontologies and Legal Information Systems. Second International Workshop on Legal Ontologies, JURIX 2001, Amsterdam.
2. Didier Bourigault, Guiraudé Lame, Analyse distributionnelle et structuration de terminologie - Application à la construction d'une ontologie documentaire du Droit, *Revue Traitement Automatique des Langues*, 43-1, 2002.
3. Leonardo Chiariglione, Digital Media Project, 2003 <http://www.chiariglione.org>
4. Jaime Delgado, Isabel Gallero, Standardisation of the management of Intellectual Property Rights in multimedia content, Proceedings of the Second International Conference on Web Delivery of Music, IEEE Computer Society, 2002, p. 125-132.
5. Melanie Dulong de Rosnay, Digital Right Management systems toward European law: between copyright protection and access control, Proceedings of the International Conference on Web Delivering of Music, Darmstadt, 9-11 December 2002, IEEE Computer Society Press, pp. 117-124.
6. Séverine Dussolier, Fair use by design in the European Directive of 2001: an empty promise, 12th Conference on Computers, Freedom and Privacy, 2002. <http://www.cfp2002.org/fairuse/dussollier.pdf>
7. Tom Gruber, A Translation Approach to Portable Ontology Specification, *Knowledge Acquisition*, 5/1993, pp. 199-220.

8. Guiraudé Lame, Using text analysis techniques to identify legal ontologies components, ICAIL 2003, Workshop on Legal Ontologies & Web Based Legal Information Management
9. Lawrence Lessig, Code and Other Laws of Cyberspace, Basic Books, 1999
10. Lawrence Lessig, The future of Ideas, The Fate of Commons in a Connected World, Random House, 2001.
11. Jessica Litman, Revising Copyright Law for the Information Age, 75 Oregon Law Review 19, 1996. <http://www.law.cornell.edu/commentary/intelpro/litrvtxt.htm>
12. Deirdre Mulligan, Aaron Burstein, 2002. Deirdre Mulligan, Aaron Burstein, Implementing Copyright Limitations in Rights Expression Languages, 2002 ACM Workshop on Digital Rights Management.

Legal Texts

13. Berne, 1886. Berne Convention for the Protection of Literary and Artistic Works <http://www.wipo.int/clea/docs/en/wo/wo001fr.htm>
14. Directive, 2001. Directive 2001/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonisation of certain aspects of copyright and related rights in the information society, OJ L 167 22.06.2001 p. 10. <http://www.europa.eu.int/eur-lex/en/index.html>
15. US Copyright Law. Copyright Law of the United States. <http://www.copyright.gov/title17/index.html>
16. DMCA, 1998. Digital Millennium Copyright Act. <http://www.loc.gov/copyright/legislation/dmca.pdf>
17. WCT, 1996. WIPO Copyright Treaty, World Intellectual Property Organization, 20/12/1996 <http://www.wipo.org/eng/diplconf/distrib/94dc.htm>
18. WPPT, 1996. WIPO Performances and Phonograms Treaty, World Intellectual Property Organization, 20/12/1996. <http://www.wipo.org/eng/diplconf/distrib/95dc.htm>

DRM Standardization

19. MPEG, <http://www.mpeg.telecomitalia.com>
20. MPEG-21 <http://www.chiariglione.org/mpeg/index.htm>
21. MPEG-21 N4518. ISO/IEC JTC 1/SC 29/WG 11 N4518, From MPEG-1 to MPEG-21: creating an interoperable multimedia infrastructure <http://mpeg.nist.gov/>
22. OASIS <http://www.oasis-open.org/home/index.php>
23. Open Mobile Alliance <http://www.openmobilealliance.org/>
24. IEEE Learning Technology Standards Committee <http://ltsc.ieee.org/>
25. Internet Streaming Media Alliance <http://www.isma.tv/home>

REL/RDD Initiatives

26. Indecs <http://www.indecs.org/pdf/framework.pdf>
27. Imprimatur project <http://www.imprimatur.net>
28. ODRL <http://odrl.net/>
29. IPR System, <http://www.iprsystems.com>
30. XrML <http://www.xrml.org/>
31. Content Guard <http://www.contentguard.com>
32. IPRonto <http://dmag.upf.es/ontologies/ipronto/index.html>
33. DMAG (Distributed Multimedia Applications Group), Universitat Pompeu Fabra, Spain <http://dmag.upf.es>
34. Creative Commons <http://www.creativecommons.org/>

A Comparison of a Regulatory Ontology with Existing Legal Ontology Frameworks

John Kingston and Wim Vandenberghe

Joseph Bell Centre for Forensic Statistics & Legal Reasoning
University of Edinburgh, United Kingdom
www.josephbell.org

Abstract. In this paper we describe the nature of a regulatory ontology to be developed to support systems that tackle financial fraud. This work is part of the FF POIROT European IST project. We describe existing legal ontologies and examine then how these can be re-used to realize the ontology requirements identified for FF POIROT. We will discuss the proposed categories and their limitations for the ontological engineering of FF POIROT.

1 Introduction

The Joseph Bell Centre is a participant in the EU-funded FF POIROT project¹, which is developing an ontology for financial fraud, with a specific focus on two areas of financial fraud: unauthorised online investment solicitation, and VAT fraud. The recently completed user requirements for the ontology identified eleven categories (with several sub-categories) of concepts that need to be represented in the ontology.

The purpose of this paper is to compare these eleven categories with the recommendations made by a number of proposed legal ontologies. The ontologies that will be considered are Valente's functional ontology of law; the frame-based ontology of law proposed by van Kralingen and Visser; and the LRI-core legal ontology proposed by Breuker and others.

¹ This work was supported under the IST project FF POIROT (Financial Fraud Prevention Oriented Information Resources using Ontology Technology), which is sponsored by the European Union's Information and Systems Technology Directorate (5th framework) under grant number IST-2001-38248. The FF POIROT consortium comprises Starlab from the Free University of Brussels; the Commissione Nazionale per la Societa' e la Borsa; CVC from the Erasmus Hogeschool; the Joseph Bell Centre for Forensic Statistics & Legal Reasoning from the University of Edinburgh; Knowledge Stones S.p.A.; Language and Computing N.V.; the Romanian Academy Research Centre on Artificial Intelligence; and VAT Applications N.V. The EU IST and the partners comprising the FF POIROT consortium are authorised to reproduce and distribute reprints for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the author(s) and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of the EU IST or any other member of the FF POIROT consortium.

The format of the paper is as follows: the first section describes the user requirements identified for the FF POIROT ontology of financial fraud; the second section outlines the three legal ontologies that we will be considering (with acknowledgements to [1]); and the final section then discusses each of the ontology requirements in detail, showing how that requirement might be realised in each of the ontologies.

2 Projected Users and Domain Description

It is estimated that the EU loses several million euro per year due to financial fraud. Therefore it should come as no surprise that prevention and early detection of fraudulent activity is an increasingly important goal for the EU and its Member States.² This section will illustrate the added value that our proposed ontology for financial fraud would have for respectively a financial professional and a law enforcer in Europe, both of whom are members of the FF POIROT consortium.

2.1 VAT@

The first user envisaged for the POIROT ontology of VAT fraud is VAT Applications BV (VAT@). VAT@ is a Belgian software company developing automated software to deal with issues surrounding value added tax at a European and international level. Its primary goal is to ensure compliance with VAT regulations by cross-border traders in Europe; it also has a strong interest in helping with identification of VAT fraud.

Complying with VAT regulations requires knowledge of a vast web of regulations, and is therefore a heavily knowledge-based task. *Detecting* VAT fraud appears to require much information but comparatively little knowledge; once all the information is available, a numerical comparison between transaction sizes or identification codes on invoices, or a failure to match one invoice with another, will reveal the fraud. The European Community has developed a cross-border database (known as VIES, the VAT Information Exchange System).³ The problem is in making the information available in a timely fashion. The current operation of VIES does not allow sufficiently early availability of such information.

However, the task of fraud *prevention* – using profiles of typical fraudsters or typical fraudulent transactions to direct inspectors towards possible frauds at an early stage – appears to be an area much richer in knowledge. Profiles will consist of a mixture of single facts that raise “red flags”, conjunctions of facts that raise “red flags” if found together, and sequences of data over time (behaviour patterns).

² See Communication from the Commission, Protecting the Communities’ Financial Interests. Fight Against Fraud. Action Plan for 2001-2003, COM(2001) 254 final. A recent effort by the UK government is the initiative by HM Customs & Excise, titled “Protecting Indirect Tax Revenues”, designed to save £2 billion per year.

³ Certain parts of this common computer network can be accessed by businesses across Europe to check the validity of their customers’ VAT identification number on the Internet since 14 June 2002. See http://europa.eu.int/comm/taxation_customs/vies/en/vieshome.htm

Currently, the frequency of control visits is primarily determined by the size of the enterprise, with some attention being paid to other risk factors such as the perceived risk of individual traders, whether the trader is dealing in a fraud-sensitive branch or whether it is a newly founded business. A better understanding of fraud-related risk factors could lead to better targeting of control visits and a significant increase in fraud prevention. It may be that this area is considered more suitable for an ontology-supported demonstrator than the task of fraud detection.

The ideal system for supporting VAT compliance would be a knowledge based system that asks the user a few questions (in a choice of languages) and then supplies all the necessary information and forms for VAT compliance. If such a system were to be supported by an ontology, the requirements on the ontology would be:

- To represent legislation and legal rules from multiple countries;
- To support reasoning about compliance with these legal rules;
- To associate natural language terms in various languages with key concepts, so that (for example) key rules from one country can be viewed in the language of another country.;
- To represent interpretation of legislation and legal rules useful to the user.⁴
- To maintain an audit trail – this is critical for compliance

Specific requirements for a system that supported VAT prevention would include:

- The system should be capable of identifying indicators of fraud on invoices, including multiple pattern indicators (i.e. combinations of factors that raise no suspicion if seen in isolation) and distinguishing fraud-related patterns from simple irregularities.
- The system must deal with multiple data sources (VIES, ICT listing, etc), each having different functions and user interfaces.
- The system should know about law related to fraud investigation as well as fraud identification. For example, Customs & Excise in the UK have power of criminal investigation, but only the police such powers in Belgium.
- The system should automate repetitive tasks such as checking to ensure VAT numbers are valid.

2.2 Consob

The Commissione Nazionale per le Società e la Borsa (Consob) is the public authority responsible for regulating the Italian securities market. Its activity is aimed at the protection of the investing public.

For this project, the focus will be on websites offering online investment services to Italian residents and abusive solicitation of public savings. Websites may be identified as unauthorized because they have failed to comply with the rules for

⁴ For example, HM C&E have refused to zero-rate supplies from companies which were complying with UK VAT law (notice 703). See, Case Study “How HM Customs & Excise misinterprets EC & UK VAT law to the detriment of British business trading in Europe”, <http://www.bavat.co.uk/cases/ec.htm>

authorization, and/or because they contain specific indicators of fraud – for example, offering plausible-sounding but non-existent financial products.

Consob's current procedure for finding non-compliant or fraudulent websites is fundamentally based on keyword-search, consisting of using different Internet search engines (such as AltaVista, Google, Yahoo, etc) as well as several meta-search engines and neural networks. The search result is a list of web sites whose content is investigated by Consob's inspection officers in order to analyse and identify market abuse phenomena, abuse provisions of investment services and investment solicitation. The keywords are selected and combined manually to create complex queries on basis of the experience acquired by Consob's operative units.

Within the FF POIROT project, Consob is hoping to automate and update its procedure to spot frauds more efficiently. The ultimate goal of Consob is to use the FF POIROT ontology with software tools to automate the launching of queries that will find suspect websites and to optimise the web information retrieval results. Key requirements include:

- The ability to examine possible fraudulent websites, and the links included on that website;
- The ability to use several subject-specific thesauri -- databases of term phrases with respect to the specific crime of fraudulent online investment services to Italian residents and abusive solicitation of public savings;
- The ability to find suspicious information on the World Wide Web, using tools that are able to determine which sites are offering fraudulent online investment services to Italian residents. This requires:
 - being able to identify web-based operators offering investment services or soliciting public savings;
 - being able to identify those operators who are not licensed;
 - being able to prove that these operators are engaged in offering investment services or soliciting public savings.
- The ability to assign weights of suspicion to different websites.
- The ability to organize and manage search results.
- The ability to support a natural language processing tool that is able to conduct semantic analysis of the selected pages to identify the pages containing crime information. Furthermore, if a page is verified as containing crime information (by the tool or by a human), it will be processed automatically to abstract new concept terms that are to be added to the database for supporting further search.
- To support a central repository to store the relevant web sites after the semantic analysis. The system should keep a detailed history of the fraudulent website and changes to it to make the job easier for prosecutors.

3 User Requirements for an Ontology of Financial Fraud

The ontology requirements were designed to support these regulatory concepts:

- The law regarding authorised investment solicitation and VAT transactions;
- Actions to be taken when these laws are breached;

- Objects and concepts involved in online investment solicitation;
- Objects and concepts involved in VAT compliance and VAT fraud (which are considered to be different, since fraud implies a deliberate plan to make money and to appear to comply with regulations for as long as possible, so detecting non-compliance and detecting fraud are very different activities);
- Objects and concepts involved in the detection of breaches of these laws.

The user requirements that were identified are as follows:

1. Representing national and supranational law regarding online investment solicitation and VAT. This requirement was broken down into several levels of detail; legal rules were decomposed into postulates, facts, and evidence that might match those facts, according to the epistemic framework of Wigmore [2], while the (hypothesised) plans and goals of fraudsters must also be represented;
2. Representing law relevant to the investigation of fraud. This included law regarding enforcement authority and capabilities; laws relating to seizure of evidence; and information exchange mechanisms between enforcement agencies;
3. Representing products or commodities, including financial products;
4. Representing commercial transactions, in terms of a sale, a vendor & a purchaser.
5. Representing VAT invoices: a special case of a sale record, given its own category because it is so important in the detection of VAT non-compliance;
6. Representing indicators of fraud;
7. Representing companies and their structure;
8. Representing individuals and their relationships (on the assumption that criminals tend to operate in networks);
9. Representing databases and their communication formats (a special case of representing information exchange mechanisms);
10. Representing websites;
11. Representing thesaurus/alternative natural language terms for the same concept, to a level where they can be used as search keywords.

4 Legal Ontologies: A Brief Description

A legal ontology is a kind of a proto-theory which indicates concepts that exist within the legal domain and how these concepts are related to one another. Legal core ontologies consist of concepts that are general for all legal domains. Concepts being captured by such legal core ontologies include agent, role, intention, document, norm, right, and responsibility. Thus, a legal core ontology mediates between a foundational (reference or upper-level) ontology and specific legal domain ontologies [3].

A small number of legal core ontologies is available⁵, and we will review three of them here. Valente's *functional ontology of law* [4], obviously, adopts a *functional* perspective of law (FOLaw). The main function of a legal system is to regulate social behaviour (law as a social control system). By using this functional view of law, categories of legal knowledge are distinguished which are represented in the ontology. Valente distinguishes six categories of legal knowledge: (a) normative knowledge; (b)

⁵ For an overview, see [1].

responsibility knowledge; (c) world knowledge; (d) reactive knowledge; (e) meta-legal knowledge; and (f) creative knowledge.

- (a) *Normative knowledge* defines a standard of social behaviour. It thereby sets down behaviour of the people in the society. The standard is defined by issuing individual norms, expressing what ought to be the (compliant) case.
- (b) *Responsibility knowledge* is legal knowledge that either extends or curtails the responsibility of an agent for its behaviour. Its function is to provide the legal means to reject the common idea that someone is only responsible for what one causes. Prime examples of this ‘fault-causation-responsibility’ problem are to find in tort law and fraud cases (see below).
- (c) *World knowledge* is legal knowledge that describes the world that is being regulated. It delineates the possible behaviour of persons and institutions in a society. Thereby it provides a framework to define what behaviour ought (and ought not) to be performed.
- (d) *Reactive knowledge* is legal knowledge that specifies which reaction should be taken (and how) if an agent violates a primary norm. Usually, this reaction is a penalty (fine and/or imprisonment, etc).
- (e) *Metalegal knowledge* is legal knowledge about legal knowledge, or, legal knowledge that refers to other legal knowledge. It deals with legal principles in case of conflict of norms (often Roman law adagia).
- (f) *Creative knowledge* is legal knowledge that allows the creation of previously non-existent legal agents, bodies and entities (e.g. law enforcement agency, a contractual agreement).

Apart from the world knowledge category, which has an implied ontological foundation, all categories are mainly concerned with the epistemological realm [5].

Key distinctions made include:

- Division of responsibility into ‘legal responsibility’ & ‘causal responsibility’
- The distinction between norms that *command* behaviour, norms that *empower* an agent, and norms that *derogate* some other norm (a special case of which is the *permitting* of some behaviour);
- The division of world knowledge into *definitional* knowledge and *causal* knowledge. Definitional knowledge consists of legal concepts, legal relations, a specific case (i.e. problem case being investigated), circumstances (i.e. facts of a case), and generic cases (typical case). Causal knowledge establishes causal links between agents and states.

Another legal core epistemology is the ontology developed by [6]. This *frame-based ontology of law* (FBO) is decomposed into the generic legal ontology (norms, acts, concept descriptions) and the statute-specific ontology.

The former divides legal knowledge over *norms* (rules, standards and principle of behaviour), *acts* (“dynamic aspects which effect changes in the state of the world”), and *concept descriptions*. A statute-specific ontology is also proposed as an instantiation of the generic one, to deal with the fine details of individual cases.

Key distinctions made by van Kralingen and Visser include:

- A distinction between physical acts and institutional acts (legal interpretations of physical acts that occur in the real world);

- An emphasis on actions' modality (e.g. using a weapon; aggressively), temporal and spatial aspects (e.g. on foggy nights; in the East End of London), and circumstances (e.g. during wartime).
- Representing the scope, conditions of application, and legal modality (ought, ought not, may, can) of norms.

The *LRI core ontology* [7], [8] is being developed in the context of the E-court and E-power projects.⁶ It includes Valente's FOLaw functional ontology, but differs from it by being a more generic ontology with reference to legal roles, procedures, documentation, communication and legal sources. In doing so, it not only encompasses substantive law (as FOLaw already did), but also procedural law.

LRI-Core considers objects and processes to be the primary entities of the physical world, so that objects participate in processes, while processes transfer or transform energy. Mental entities are considered to behave in an analogous manner to physical objects. So facts (mental objects) exist and are stored in memory, and these facts may or may not be believed (mental process). The mental and physical world overlap in agents, and in time and space. Social organization and processes are composed of roles that are performed by agents; the law associates norms with these roles.

Key distinctions made by LRI-Core include:

- Distinction between events and processes: events are explained by processes;
- Clear definition of the role of force or energy in processes, as opposed to distinguishing objects as endurants or perdurants (i.e. whether objects can lose their identity over time).
- Emphasis on representing agents and communication between them.

Although the ontological categories adopted in the above legal core ontologies are all based on widely acknowledged views in legal theory, this does not guarantee the availability of all layers and categories needed for domain specific ontologies.

Another approach to classifying legal knowledge that should be mentioned is the document mark-up approach of *LegalXML* and *LeXML*. These are metadata standards for tagging items in documents, designed to allow consistent capture and exchange of electronic legal documents. LegalXML's focus is on the USA with its common law 'code', while LeXML concentrates on continental Europe and its civil law tradition. These are not being considered as a basis for FF POIROT, partly because Legal XML and LeXML are not yet completely defined, but mostly because the domain of FF POIROT is not one where the incoming information is likely to consist of marked-up documents. However, it is possible that the output of POIROT could consist of marked up documents – for example, if the VAT compliance route is followed, VAT invoices could be automatically marked up using Legal XML or LeXML based tags.

The next section will consider the applicability of the proposed ontologies (FOLaw, FBO, and LRI Core) to the knowledge categories of importance to financial fraud ontology (captured by the user requirements analysis). In doing so, it will pay attention in particular to the re-usability of the above discussed ontological layers.

⁶ E-court (IST-2000-28199), <http://www.intrasoft-intl.com/e-court>. E-power (IST-2000-281250), <http://www.lri.jur.uva.nl/research/epower.html>

5 Comparing User Requirements with Proposed Legal Ontologies

This section will consider how the eleven requirements identified for FF POIROT might be realised in the four legal ontologies described above.

5.1 Representing National and Supranational Law Regarding Online Investments and VAT

At an abstract level, legal reasoning according to Wigmore involves answering a number of questions about a situation, such as:

- What is the ultimate intended aim (proposition to be proven, factum probandum or hypothesis)?
- What is the substantive law that will be breached if the Ultimate intended aim is reached?
- What are the acts or omissions that need to be undertaken (or not undertaken) if the ultimate intended aim is to be achieved?
- What acts or omissions are generally seen if the ultimate intended aim is to be achieved?
- What acts or omissions are generally not seen if the ultimate intended aim is to be achieved?

Van Kralingen & Visser's top level distinction between norms, acts and concepts seems to map well to the distinctions identified in our domain between laws, fraudsters' plans & actions, and supporting facts/evidence. Furthermore, their distinction between institutional (legal) and physical acts is particularly useful in the representation of cross-border crime, since there is no single fraud offence in Europe. The physical act of defrauding someone may then be classified as any of the following institutional acts: theft; conspiracy; cheating; etc.

The proposed properties of 'norms' layer in Van Kralingen & Visser's FBO ontology are also useful for the legal fraud domain and its abundance of legal statutes and regulations, as are the properties of concepts, which allow determination of necessary and sufficient conditions – which maps well to the Wigmorean method used in the FF POIROT project. However, the two norm types (norm of conduct or norm of competence) are not sufficient to distinguish the wide array of EU law and its implementation by Member States in the fraud domain; for these laws differ in hierarchy (primary EU law, secondary EU law, national legislation, etc), applicability and in character (statutes, guidelines, etc) as well as in content.

Fraud cases typically involving multiple agents, often with multiple roles. There is the organizer, the co-organizer, 'puppets', etc. Not all of them will be held accountable to the same extent. Valente's FOLaw ontology is capable of representing this diversity through its emphasis on agents; it is able to represent agents' responsibility, causation by agents, sanctions to be applied to agents who breach laws, and the creation of new agents. The last point is critical to a domain where agents may be discovered in the course of an investigation, or even hypothesised without ever being fully instantiated. Unfortunately, agents' actions and intentions can only be represented in a rather roundabout manner: actions must be represented as causal

knowledge, and intentions by identifying that the agent is legally responsible as well as causally responsible. It may be that Valente considered that actions were already sufficiently well covered by existing ontologies of AI planning, but it would have been helpful if actions and intentions had been made more explicit in the FOLaw ontology. FF POIROT not only needs to represent fraudulent acts by fraudulent agents, but also acts by legal enforcement agents combating these fraudulent acts. The ontology is very capable of representing legal rules, however, with its helpful subdivision of norms into commanding norms, empowering norms and derogating/permitting norms.

The LRI-Core ontology is still under development, and is more general than the other ontologies studied here. However, the distinctions between mental objects and mental processes, with agents as physical and mental objects who have roles, and law as norms applied to roles, allows the FF POIROT project to represent fraud plans and intentions, actions to implement these plans, fraudsters as performers of these plans, and laws that apply to them. Even the motive of a fraudster can be represented via the energetic process category. Despite the fact that LRI-Core's notion of a legal role (judge, defence-counsellor, clerk-of-court, etc) differs somewhat from the roles considered by FF POIROT (perpetrator, accomplice, actors unknowingly part of fraud, victim and investigators), it seems that LRI-Core is strong in the areas where FOLaw is weak.

In FBO, the concepts of 'aim of an action', 'cause for an action' and 'intentionality' allow the FF POIROT project to represent correspondently fraud goals, fraud motives and the intention of the actor to commit fraud. Another strength of FBO is that it is able to represent the results and consequences of an action ('final state'). For the FF POIROT project, it is important that states of the world that result from actions can be represented, because of the need for pattern recognition (of indicators of fraud) or because of the need to consider different states of the world at different points in time in order to conclude that fraud has been committed. In FBO, LRI-core and FOLaw temporal aspects of an act can be represented. From a legal viewpoint, there are three (relative) time intervals which are important: before, during and after the (fraud) action. These time intervals can be expressed in following 'act-mode': preparatory acts or omissions, *actus reus* and consequential acts. From this survey, it's not clear whether all the ontologies can support this level of temporal detail.

5.2 Representing Law Relevant to the Investigation of Fraud

There are many different types of organizations that conduct fraud investigations. Police and government agencies have specialist fraud investigators and analysts, but forensic accountants and auditors and some specialist private organisations also conduct fraud investigation. Furthermore, as in most investigations, fraud investigations generate a huge amount of information. An ontology must therefore be able to represent laws and procedures relating to correct handling of evidence, investigative processes, entities to be investigated, and the various authorities involved in the investigation.

LRI-Core's central category of procedural law (conceptualized by the LRI developers as 'formal law') is useful to conceptualize the criminal procedural law in

investigating financial fraud. It knows about ‘hierarchy of authority’ which is important if one has to represent (e.g.) who gives the authority to obtain information and access to premises and records in fraud cases. LRI is also capable of representing the structure of an organization, e.g. fraud department (so-called ‘has-as-part social roles’).

FOLaw and FBO are suited to representing some other aspects of these ontological areas. FOLaw’s category of “Reactive knowledge” is highly suitable for representing laws regarding investigation and prosecution of fraud, while FBO’s emphasis on modality is useful to describe how investigative procedures should be carried out.

5.3 Representing Products or Commodities

Products or commodities consist of physical objects (goods) or conceptual objects (such as many financial products) that may be offered for sale. Since products are primarily financial rather than legal concepts (though they may be involved in legal concepts, if their ownership changes or is disputed, or if they are discovered to be not fit for purpose), they are not directly addressed by the four legal ontologies. However, the concept descriptions suggested by FBO include categories for the “range of application” of a concept description and for the “conditions under which a concept is applicable”, which is useful in identifying if a particular product is only useful or only available in a particular geographical area (e.g. Italy) or temporal span (e.g. financial futures). The other ontologies only consider products as peripheral concepts: in FOLaw, they may constitute the circumstances of a case (a sub-sub-category of world knowledge), while in LRI-Core, they are objects that participate in processes.

5.4 Representing Commercial Transactions

Commercial transactions require a sale, a vendor and a purchaser. A sale is an agreement between two legal entities for the exchange of a product for a price. The legal entities play the (usually distinct) roles of vendor and customer. Normally the product is a good or service and the sale price is monetary. It is also normal for a record of the sale to be created, usually on some kind of invoice.

Commercial transactions are a legal process that is carried out by (normally) two agents. As such, aspects of commercial transactions are captured by all three ontologies. FOLaw is capable of representing the responsibilities of the agents as well as the fact that the entire concept of a transaction can be considered to be a societal norm; FBO represents both norms and acts in some detail; while LRI-Core can represent the agents, the process, and the communication that takes place between agents.

5.5 Representing VAT Invoices

This knowledge category is only applicable in the area of VAT fraud. Ontologically, it belongs to the knowledge category of ‘commercial transactions’. However, since

the features that appear on an actual invoice are extremely important in both the VAT compliance domain and the VAT fraud detection domain, they constitute a separate knowledge layer.

The concept of ‘VAT invoice’ has many attributes, which might be found in other knowledge categories (e.g. company name, price, product, etc). The fields of a VAT invoice which are of importance will be represented using a consortium partner’s current technology.⁷

The FBO ontology is perhaps the most appropriate for representing VAT invoices, given the detailed set of properties that it proposes for concepts. For FOLaw, such an invoice would fall under the rather too general category of World knowledge, while in LRI-Core the emphasis would be on an invoice as a communication between agents.

Note that mathematical operations will be needed to verify the correctness of the variables in the VAT invoice attributes.

5.6 Representing Indicators of Fraud

Fraud indicators or red flags are crucial in fraud detection. They can be defined as “something different from what is expected as to time, place, personality or amount, whether ‘too big, too small, too little, too much’” [9]. Indicators will be types of evidence that can indicate a risk of fraud. This is a similar concept to the concept of ‘evidence’, except that indicators contribute to identifying fraud plans and not legal rules.

Consequently, indicators must be represented ontologically in two ways; as facts which can be instantiated by evidence, and as combinations of facts that may indicate a particular conclusion. These two concepts can be represented as ‘norms’ which are present in all three of the ontologies under discussion; however, the need to note that these are norms of bad behaviour rather than good behaviour, and the fact that they are probabilistic, is probably easier to capture in a detailed ontology such as FBO rather than a more general ontology such as LRI-Core.

5.7 Representing Companies and Their Structure

The representation of companies and their structure is important because VAT fraud often involves newly-created (or newly-vanished) companies, “buffer” companies, and chains of suppliers. These relationships can be captured by a small variation on LRI-Core’s emphasis on agents and communication between them. LRI-core’s ability to represent companies as a legal entity or person is also useful.

For FOLaw, companies fall under the category of “world knowledge”; in FBO, companies must be represented as concepts, and structure as relationships between concepts or (possibly) as scoping restrictions on concepts.

⁷ Samples are already produced by iVAT assistant, which is developed by VAT Applications BV.

5.8 Representing Individuals and Their Relationships

Individual's relationships are even more complicated to capture than company links, because individuals may be linked to others through family relationships, friendships, membership of the same organization, or regular working relations. The ontological requirements are therefore similar to those required to represent companies and their structure, but with an additional need to represent relationships and associations among agents of different types. LRI-Core's emphasis on agents and de-emphasis of static relationships should allow representation of relationships between different types of agents relatively smoothly; the division between dynamic and static relationships in FOLaw may also be useful.

5.9 Representing Databases and Their Communication Formats

In cross-border complex crime cases, investigators rely heavily on databases of information. The FF POIROT ontology must be capable of conceptualizing the database – that is, describing its content, its fields, its capacity, and importantly its communication formats, in order to be able to gather relevant information.

The representation of databases is one step removed from the worlds of finance and law – it is really an issue of computer science. None of our three ontologies would be expected to represent these concepts well. However, LRI-Core's emphasis on communication formats means that it does at least have a chance of representing some of the key information.

5.10 Representing Websites

Evidence acquisition and preservation from the WWW is an essential step in online investment fraud investigation. Besides the conceptualization of the admissibility and reliability of virtual evidence (see above, 4.2), it is necessary to represent the websites in the ontology. In addition to the content of the website, also changes in these websites over time need to be represented. Date-and-time stamps of the website need to be demonstrated.

Essentially, this ontology requirement is for the representation of a document which happens to be stored electronically. Documents can be represented as world knowledge in FOLaw, as concepts in FBO (with appropriately instantiated properties), and as communications between agents in LRI-Core; and temporal representations have been briefly discussed above. But representing a website also needs to represent following components: title, body text, hyperlinks, and perhaps other components (menus, graphics, logos, advertising, etc.). A number of these components will correspond with other parts of the ontology (e.g. the name of the company, individuals, offered product, etc.). Fortunately, there is no strong requirement to represent the layout of the website, as this is a difficult concept to represent in any ontology.

5.11 Representing Thesaurus/Alternative Natural Language Terms

The ontology needs to support the association of natural language terms in various languages with key concepts, so that, for example, key rules from one country can be viewed in the language of another country. Further, the ontology must be able to cope with synonymy, near-synonymy and multilingual representation. For example, in the UK VAT practitioners/officers do not use the word 'exempted' but use instead 'zero-rated' to denote if a product is exempted from VAT. Using 'zero-rated' in communications with non UK VAT practitioners/officers might create confusion.

Related to this – but in the online investment fraud domain – are URL names that are very much like the URLs of real financial institutions. It is well known that fraudsters copy the URL names to attract (absent-minded) WWW surfers.

Further, the ontology must be able to represent words or phrases that are considered to be suspicious if found in certain contexts. In the online investment domain, for example, there are often clues when websites contain terms that either do not exist in financial circles or which are derivations of actual terms.

As with the representation of databases, the representation of natural language terms is one step removed from the financial & legal worlds. However, the terms themselves are very much part of these worlds, and may well appear in relevant databases, so these two requirements are in fact related. The need to represent natural language terms as concepts that must be searched for places a strong requirement on ontologies – or rather, implementations of ontologies – to be flexible enough for identifying labels of concepts to become concepts in their own right. It's not clear how well the three ontologies considered here could support this requirement, although FOLaw's ability to create new concepts seems promising.

6 Discussion

Financial fraud investigation involves a wide range of measures in several sub-domains. This involves international finance, accountancy, tax law, police procedure and evidence handling, general legal and law enforcement knowledge, comparative law knowledge, and knowledge of databases and linguistic descriptions. As a result, the FF POIROT ontology is not exclusively confined to the legal domain: it needs to draw from ontologies of laws, of evidence, of finance, and even of computer science.

This should explain some of the shortcomings of the existing legal ontologies for supporting FF POIROT. However, there is real value in using one of these legal ontologies for the purposes of standardisation, to provide guidance on concepts that ought to be represented, and to provide a representation that is based on a well-founded theory and hence is robust to future extensions. Furthermore, each of these legal ontologies is situated within a wider ontology framework, and perhaps the non-legal aspects of the required ontology can be obtained from other areas of the respective ontology framework.

Considering the three ontologies that were examined in this study, it seems that laws or legal rules can be represented adequately in any of them. It is in the associated areas that the most differences appear: FOLaw's functional perspective is useful for dynamic situations where new agents or concepts are being created (such as

investigations), FBO is particularly useful for representing static items that have many finely detailed features (such as VAT invoices), and LRI-Core is particularly useful when considering agents and communication between them (e.g. representing individuals and their relationships). Perhaps the fundamental message of this study is that real world problems combine dynamic and static situations, inanimate objects and relationship networks, and law with other disciplines; and so flexibility in ontological representation is to be highly valued when selecting an ontology for such a project.

Certain issues also raise their heads in a real-world problem such which may cause difficulties for ontological representation. Issues identified so far include:

- In the field of law dealing with new technologies (such as the Internet), it is important to remember that new concepts appear regularly. This is a result of the government issuing new regulations and norms. New interpretations of old concepts may also arise.
- Fraudsters change their procedures frequently in order to avoid detection. Any fraud plans and goals that are represented should therefore (ideally) be modular so that they can be re-assembled in a different manner to represent a new *modus operandi* of fraud.
- The level of detail required to represent different types of fraud is different (for example, invoices must be represented in detail for identifying VAT fraud but not for unauthorised online investment solicitation) which limits ontology re-usability.

References

1. GANGEMI, A., GUARINO, N. and DOERR, M.: "Harmonisation Perspectives of Some Promising Content Standards", OntoWeb Consortium (2002)
2. WIGMORE, J.H.: *The science of judicial proof: as given by logic, psychology, and general experience and illustrated in judicial trials*, 3rd ed. Boston: Little, Brown (1937)
3. VALENTE, A., BREUKER, J.A. and BROUWER, P.W.: "Legal Modeling and automated reasoning with ON-LINE", *International Journal of Human Computer Studies* (1999), 51, 1079–1126
4. VALENTE, A. and BREUKER, J.: "A Functional View of Law", in BARGELLINI, G. and BINAZZI, S. (eds), *Towards a global expert system in law*, CEDAM Publishers, Padua, Italy (1994)
5. BREUKER, J.A., ELHAG, A., PETKOV, E. and WINKELS, R.: "IT Support for the Judiciary: Use of Ontologies in the e-Court Project" (2002)
6. VAN KRALINGEN, R.W.: "A Conceptual Frame-based Ontology for Law, *First International Workshop on Legal Ontologies*, University of Melbourne (1997)
7. BREUKER, J.A. and WINKELS, R.: "Use and Reuse of Legal Ontologies in Knowledge Engineering and Information Management", *ICAIL Workshop on Legal Ontologies* (2003)
8. BREUKER, J.A. and BOER, A.: "Developing Ontologies for Legal Information Serving and Management", *Proceedings of the EKAW Workshop on Knowledge Management through Corporate Semantics Webs* (2002)
9. MOLLOY, T.: "Red Flags and Patient Questioning", *Fraud Intelligence* 2001, (41) 3

Representing Multilingual and Culture-Specific Knowledge in a VAT Regulatory Ontology: Support from the Termontography Method

Koen Kerremans, Rita Temmerman, and Jose Tummerts

Department of Applied Linguistics – Centrum voor Vaktaal en Communicatie (CVC)

Erasmushogeschool Brussel

Trierstraat 84, B-1040 Brussels, Belgium

{koen.kerremans,rita.temmerman,jose.tummerts}@ehb.be

<http://cvc.ehb.be>

Abstract. We intend to reflect on the problems encountered and methods applied when trying to represent multilingual and culture-specific knowledge in a terminological database which is set up to provide multilingual input for ontology engineers. The database consists of terminological information referring to units of understanding in the domain of value added tax. This database could be one of the main resources for the multilingual refinement within the DOGMA ontology engineering framework. We focus on how the method – called *termontography* – adds to the representation of multilingual and culture-specific knowledge in a European VAT regulatory ontology. What we propose is a *unit of understanding approach* which starts from a language-independent domain-specific framework arrived at with the help of domain specialists.

1 Introduction

An ontology representing the European value added tax (VAT) regulatory domain needs to account for both culture-specific as well as non-culture specific VAT *units of understanding*¹ and relationships in order to cover the VAT legislations of the different European member states. For instance, the unit of understanding (UoU) paraphrased in English as ‘VAT deduction on copyright publications’ (section 285bis

¹ The notion of unit of understanding (UoU) was introduced in sociocognitive terminology theory [1, 2] in order to clarify the inadequacy of classical concept theory for the conceptual structure of most specialised fields. Multilinguality and a multicultural setting justify the UoU approach even more (see section 4 of this paper) as the UoUs exist at a level which is independent of but can be related to the possible ways of expression in any particular human language. As in terminology theory a ‘concept’ is by definition clearly delineated and preferably referred to by one term only, we will avoid the term ‘concept’ in this article and instead distinguish between UoUs (at the highest level of abstraction) and categories which by definition allow for prototypically structured understanding, i.e. fuzziness and multiple ways of lexicalisation.

in the French VAT legislation) is culture-specific as only the VAT legislation in France makes note of it. Another example is the UoU denoting a special kind of export license, which is in Italian lexicalised as ‘esportatori abituali’, (section 8c in the Italian VAT legislation). This UoU also occurs in the French and Irish VAT legislations but does not have an equivalent in the legislations of the other European member states. These two examples show that a European VAT regulatory ontology must deal with the legislation of each European member state separately.

In this paper we intend to reflect on the methods applied and problems encountered when trying to represent multilingual and culture-specific terminological knowledge in a *database* which is set up to provide multilingual support for ontology engineers.² We propose a *UoU approach* called *termontography* [6] in which one starts from a pre-defined, language-independent framework of domain-specific UoUs and intercategory relationships, set up in collaboration with field specialists, to which lexicalisations (from a domain-specific corpus) are mapped. Lexicalisations referring to UoUs are listed in a *termontological database* – currently developed for DOGMA³ ontology engineers in the FFPOIROT project⁴ – which not only covers a language-independent description of each UoU, but also offers specifications of possible culture-specific and/or language-specific items.

Whereas a termontological database takes a UoU as a starting-point, a terminological database starts from terms in textual corpora, which are provided with descriptions in natural language, depending on the user requirements. The termontological database guarantees a better integration of the multilingual terminological information in ontology engineering frameworks.

The paper will be further structured as follows. In section 2 we will show by means of an example the difficulties in representing and aligning multilingual information. After briefly having outlined the termontography methodology (3) we will focus on the development of such a domain-specific framework (4). We will look at its particular features and show by means of an example of the VAT regulatory domain how the termontography method tries to integrate the cultural and multilingual diversity in a termontological database. Next, we will reflect on some general implications for the categorisation framework.

2 Differences in Language and Culture

The UoU ‘transaction for which no VAT is required’ is non-culture specific as it appears in the legislation of each European member state. It is even non-language specific in the sense that the paraphrase ‘transaction for which no VAT is required’,

² The idea of integrating terminological knowledge in the process of ontology engineering has already been worked out in the medical domain, e.g. the ONIONS (ONTologic Integration Of Naïve Sources) methodology [3, 4, 5].

³ DOGMA stands for ‘Developing Ontology-Guided Mediation for Agents’ [7].

⁴ (IST 2001-38248), 5th framework. The FFPOIROT project aims at compiling for several languages (Dutch, Italian, French and English) a computationally tractable and sharable knowledge repository for the financial forensics domain. For more information: see <http://www.ffpoirot.org>.

which happens to be in English, could just as well be expressed in any other natural language. Further specifying this UoU – i.e. fully relying on the expertise of the field specialist without consulting any textual material – reveals a frame of four non-culture specific, non-language specific subcategories. These subcategories, which are defined in the Sixth European Council Directive (Sixth Directive)⁵, are paraphrased in English as follows:

- subcategory 1: ‘transaction not allowing the supplier to deduct VAT’ (section 13)
- subcategory 2: ‘transaction allowing the supplier to deduct VAT’ (section 28quater or 28(c))
- subcategory 3: ‘transaction occurring outside the territory of the VAT legislation at stake’ (sections 8 and 9)
- subcategory 4: ‘transaction occurring outside the scope of VAT’ (sections 2 to 7)

The lexicalisations of the subcategories of this frame can be searched for in text corpora. They are extracted either manually or (semi-)automatically from the VAT legislation of each European member state. Difficulties arise when representing and aligning multilingual information in the termonological database. These difficulties are due to differences in language (2.1) and culture (2.2).

2.1 Differences in Language

Each VAT legislation is stated in a different language. The problem of aligning VAT legislative terminology cannot simply be solved by looking for the translation equivalents of terms as some terms are polysemous. For instance, the term ‘vrijstelling’ in the Belgian legislation is used to denote the first three subcategories of the UoU ‘transaction for which no VAT is required’. This will cause problems when one wants to know the UK translation of the Dutch term. For, if used to denote the first category, the term ‘vrijstelling’ is translated into ‘exemption’. If used for the second category, the English term ‘zero-rated’ is the only proper translation whereas for the third category, the lexicalisation ‘outside the scope of VAT’ is the only suitable translation candidate.

2.2 Differences in Culture

Another complexity arises when the same language is used in different cultural settings. For instance, in the Irish VAT legislation, the English terms ‘exemption’ and ‘zero-rated’ are used to denote the third category. In the UK VAT legislation, however, the exemption lexicalises the first category, whereas zero-rated is used for the second (see section 2.1).

The fact that cultures may perceive seemingly equivalent categories differently is shown in the following examples. Although the Dutch term ‘vrijstelling’ and the English-UK term ‘zero-rated’ both refer to transactions in which a supplier has the right to deduct VAT, it does not follow that both terms cover exactly the same list of

⁵ The Sixth Council Directive (Directive 77/388/EEC) of 17th of May 1977 deals with the harmonisation of the laws of the member states relating to turnover taxes.

possible transactions. Another clear example is the UoU referred to in English as ‘taxable event’ which is defined in article 10 of the Sixth Directive but implemented differently in the legislations of the different member states (see e.g. section 6 of the Italian VAT legislation, section 269 of the French VAT legislation or section 6(2) of the UK VAT legislation).

3 Termontography: Motivation and Methodology

The problems described in section 2 can be overcome if one has a preliminary understanding of the subject. This requires the UoU approach, called termontography, in which one starts from a pre-defined framework of categories and relationships, produced in collaboration with field specialists, to which terms and verbal constructions are mapped in a second phase.

In section 4, we will further elaborate on the features of such a categorisation framework and apply it to the example of ‘transaction for which no VAT is required’. In this section, we will briefly introduce the termontography method in order to give the reader an understanding of the workflow of which the development of the categorisation framework is part.

Termontography is a multidisciplinary approach in which theories and methods for multilingual terminological analysis of sociocognitive theory [1] are combined with methods and guidelines for ontological analysis [8, 9, 10]. The motivation for combining these two research fields derives from our view that existing methodologies in terminology compilation [11, 12, 13] and ontology development have significant commonalities. For instance, when building an ontology or compiling a terminological database, both ontologists and terminographers will start from the identification of their purposes, the restriction in the scope of the domain, the specification of the user requirements as well as the acquisition of domain knowledge.⁶

The termontography method can be summarised as follows: first of all, in close collaboration with specialists of the domain of interest, a framework of domain-specific categories and intercategory relationships is developed (see section 4). This framework facilitates the manual and semi-automatic extraction of knowledge from a textual corpus. It will gradually evolve towards an enriched and more fine-grained network of semantic relations, reflecting culture-specific conceptualisations. This analysis results in a termontological database, which is a supportive resource for the DOGMA platform [7].

This approach to multilingual knowledge representation, which is in line with the ‘object-oriented’ approach proposed by Agnesund [14], differs from attempts to create multilingual upper-level ontologies like in the MULECO-project [15], or from attempts to establish correspondences between wordnets in different languages such as EuroWordNet (EWN) [16] and MultiWordNet (MWN) [17], or from the mapping of keywords in several languages to the same concept in the ontology [18]. The UoU

⁶ Subsequent to these steps, the approaches differ as terminologists tend to focus on the representation of knowledge in natural language, whereas ontologists are concerned with a formal representation of this knowledge.

approach proposed by CVC Brussels⁷ permits one to examine which UoUs are expressed in all languages, and which ones are lexicalised in only a subset of the languages. Moreover, it reveals lexical gaps in individual languages, as well as UoUs that are particular to one language only.

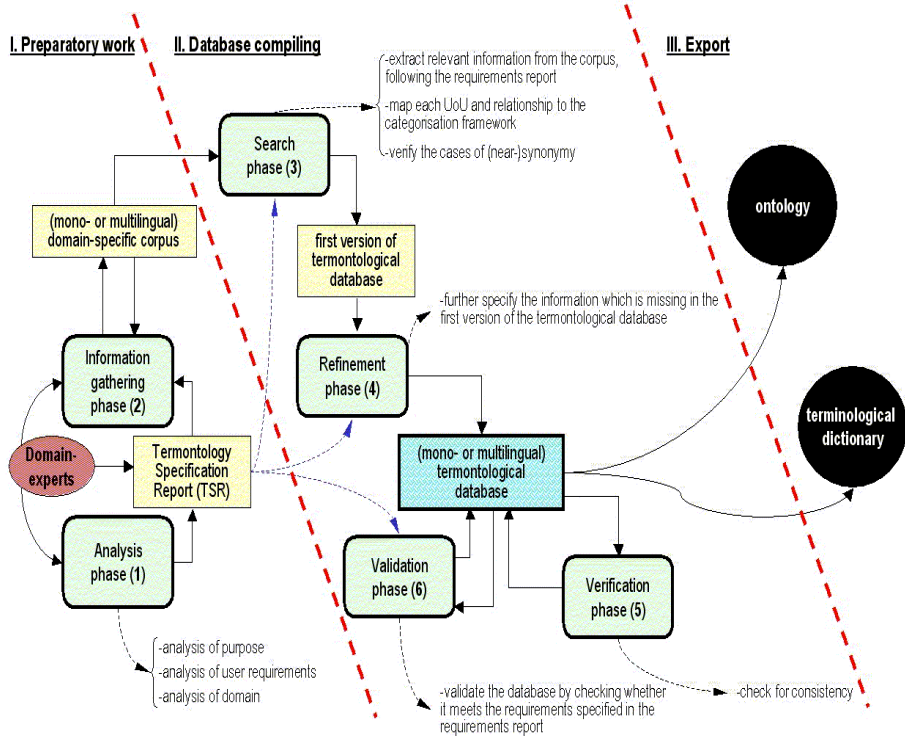


Fig. 1. The termontology workflow

Figure 1 sketches the termontology workflow, starting from the preparatory work in the analysis (3.1) and information gathering (3.2) phases, the actual database compiling consisting of the search (3.3), refinement (3.4), verification (3.5) and validation (3.6) phases to the final export of the knowledge from the mono- or multilingual termontological database, depending on the purpose of the project (see 3.1), to an ontology and a terminological dictionary. We will briefly describe each phase.

3.1 Analysis Phase

Starting from a general ontology user requirements report [10], the termontographer defines his Termontology Specification Report (TSR) in which he should find

⁷ Centrum voor Vaktaal en Communicatie Erasmushogeschool Brussels: <http://cvc.ehb.be>.

specifications concerning the purpose of the termontological database, the scope of the domain and the requirements of its users. In the VAT case, the *purpose* of the database is roughly to capture and present multilingual information about terms referring to (European) VAT legislative UoUs. The *scope* of the domain depends on the purpose of the database and is restricted to the categories that occur in the categorisation framework. The *users* are ORM modellers [19] who follow the DOGMA modelling approach [7].

3.2 Information Gathering Phase

Both the categorisation framework as well as the TSR will assist the termontographer in his search for relevant textual material with the purpose of compiling and managing a domain-specific corpus. Domain-experts play an essential role in this phase. They can point the termontographer to relevant domain-specific textual material or can recommend particular websites from which to retrieve texts.

3.3 Search Phase

From the corpus the termontographer will extract (in case of a multilingual project: for each language separately) the terms referring to domain-specific UoUs, direct them to their respective destinations in the categorisation framework and assign them a categorial label. Verbal patterns indicating intercategorial relationships will be mapped to the framework as well. This will result in a first version of a termontological database (Fig. 1).

3.4 Refinement Phase

The purpose of the refinement phase is to further complete the termontological database by for instance: aligning those terms that are equivalent (if this has not yet been done during the search phase), specifying the co-texts or concordances in which terms occur as well as the reference to the source from which each co-text was extracted.

3.5 Verification Phase

Verification refers to the process in which the termontographer checks the consistency and correctness of the termontological database. One could for instance check whether all terms have been assigned the right categorial label (section 4.1).

3.6 Validation Phase

Although to some extent being a redundant process as the TSR is constantly being addressed in practically all the methodological steps in the workflow (fig. 1), the

termontographer needs to check whether the content of the termontological database really meets the requirements as they are specified in the TSR.

Now that we have explained the important notions in the termontography workflow, we can turn to the discussion of the categorisation framework developed in the analysis phase of the termontography method (see section 3.1) that needs to reflect both culture-specific as well as non-culture specific categories of the domain of interest.

4 A Categorisation Framework

Section 4.1 deals with the particular features of the termontography categorisation framework. The framework is applied to the example of ‘transaction for which no VAT is required’ in order to show how the termontography method tries to integrate multilingual and cultural diversity in a termontological database. In section 4.2, we will reflect on some general implications of the categorisation framework.

4.1 Coping with Language-Specific and Culture-Specific Problems

The idea of first setting up and agreeing (in collaboration with field specialists) on a categorisation framework, is inspired by existing methodologies in ontology engineering [8, 9]. Terminologists like Vouros and Eumeridou also stress the need for such a categorisation framework as the starting point of all terminological work [20]. In a multilingual project like the VAT case, the categorisation framework serves as ontological backbone for both the manual and semi-automatic extraction, categorial labelling and structuring of multilingual terminology from the domain-specific corpus. Consider the example of ‘transaction for which no VAT is required’. As was clear from section 2, this UoU was further divided into four (non-culture specific) subcategories. These four subcategories as well as some of the terms that are used to denote them are listed in table 1.

Table 1. Some lexicalisations of the subcategories of ‘transaction for which no VAT is required’

	DUTCH (BELGIUM)	FRENCH (BELGIUM)	ENGLISH (UK)	ENGLISH (IRELAND)
1 st subcategory	<i>vrijstelling</i>	<i>exemption</i>	<i>exemption</i>	[to be specified]
2 nd subcategory	<i>vrijstelling</i> <i>- vrijstelling</i>	<i>exemption</i>	<i>zero-rated</i>	[to be specified]
3 rd subcategory	<i>-niet onderworpen aan BTW</i>	[to be specified]	<i>outside the scope of VAT</i>	<i>- exemption</i> <i>- zero-rated</i>
4 th subcategory	<i>niet onderworpen aan BTW</i>	[to be specified]	[to be specified]	[to be specified]

In section 2 it was said that the complexity of aligning and representing multilingual information is due to the differences in language and culture. Table 1 shows that the problems that arise when aligning the VAT legislations on the basis of lexical equivalents can be overcome by directing each term to a UoU in the

framework, thereby providing the term with a categorial label (see table 2). However, such a specification still leaves a lot of ambiguity and therefore needs further improvement. For instance, the ambiguity that arises between homographic terms from different languages, such as the English ‘exemption’ in the Irish legislation (see 3rd subcategory), is circumvented when assigning to each term, apart from the categorial label, the label of the language in which the term is used. For instance the English term would be annotated as follows: `<En><3rd subcategory>exemption</3rd subcategory></En>`.

This specification is clearly an improvement over the first one but does not yet resolve the ambiguity that arises between terms in the same language, spoken in different cultural settings. In order to distinguish the English term ‘exemption’ used in the third subcategory from the lexical correspondent used in a different culture to denote the first subcategory, a third kind of specification is required which takes into account the cultural setting of each term. With respect to the English term, this will finally result in the following specifications (respectively for English-Ireland and English-UK): `<En-Ir><3rd subcategory>exemption</3rd subcategory></En-Ir>` and `<En-UK><2nd subcategory>exemption</2nd subcategory></En-UK>`.

In the termontography method, this kind of specification is a consequence of the fact that lexicalised entries of different subcategories of the UoU ‘transaction for which no VAT is required’ are mapped to their respective subcategories (thereby receiving a categorial label), while the language and cultural setting of the texts from which they are extracted (either manually or semi-automatically) will have been specified during the information gathering phase in the workflow (see section 3).

Figure 2 shows a sample of a categorisation framework for the VAT fraud domain in which the (sub-)framework of the VAT regulatory domain will eventually be included.

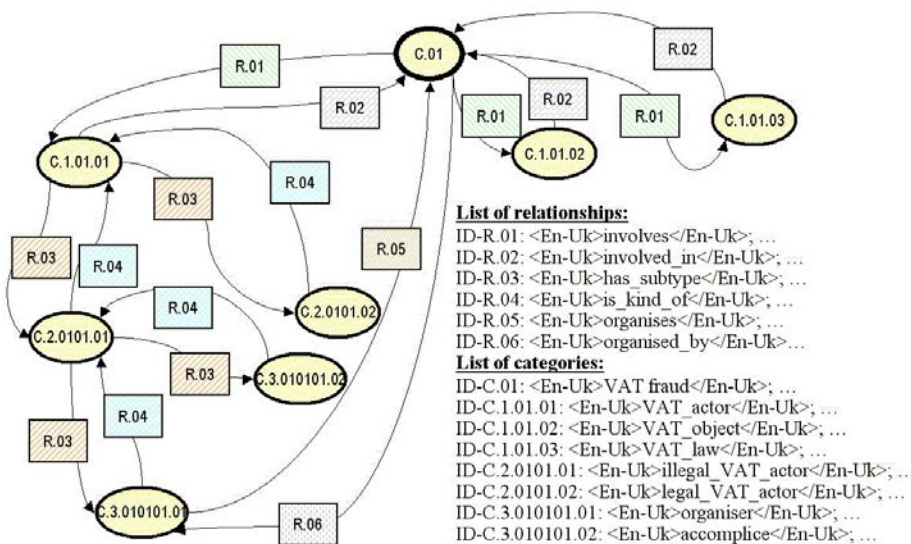


Fig. 2. A partial categorisation framework for VAT fraud

The nodes represent the UoUs in the framework, whereas the rectangles indicate the intercategorical relationships. Each UoU and relationship is assigned a unique identifier (ID). The ‘UoU ID’ is composed of the following information: the type of element (UoU versus relationship), the level of depth⁸ as well as the path one should follow, starting from the top node, in order to arrive at the identification number. The ‘relationship ID’ is only composed of information about the type of element (in this case ‘R’) and an identification number. For instance, the node ‘C.3.010101.01’ and relationship ‘R.01’ are decomposed as follows:



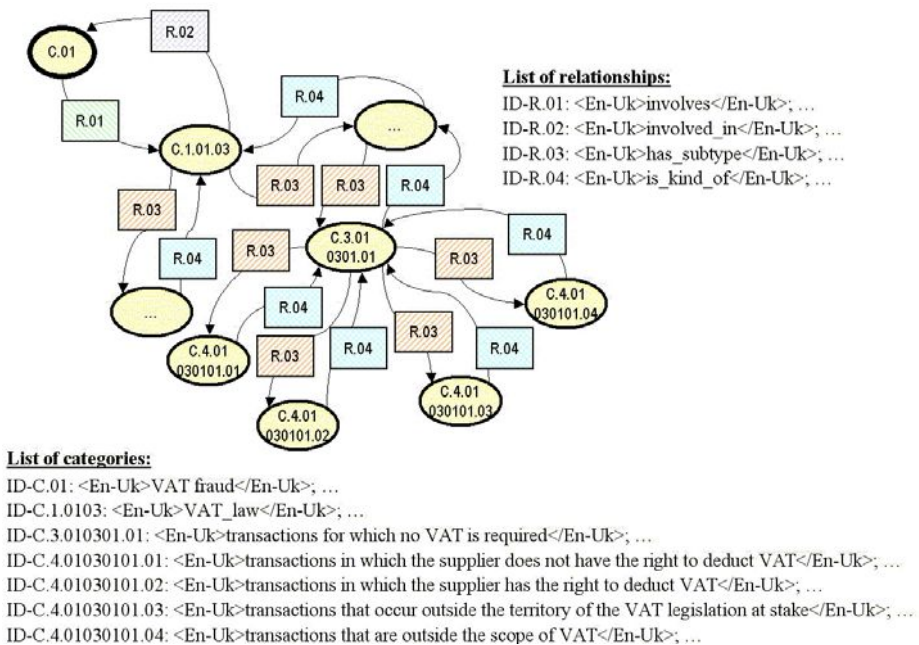
Fig. 3. Decomposing Unique Identifiers

The parent-child relationship between two UoUs, for instance ‘C.1.01.01’ (parent) and ‘C.2.0101.01’ (child), is reinforced in the example above by the relationships ‘R.03’ and ‘R.04’, respectively referring to the ‘has_subtype’ and ‘is_kind_of’ relationships. The subcategories inherit in this way the features of the parent UoU. This makes the categorisation framework already a kind of inference tree.

The node labelled ‘C.01’ and lexicalised in English as for instance ‘VAT fraud’ (English) is marked in bold which means that one should consider this particular UoU as the onset for the development of the overall framework. The relationships ‘R.01’ (‘involves’, ...) and ‘R.02’ (‘involved_in’, ...) link this particular UoU to its directly associated UoUs which restrict the domain of VAT fraud to a particular scope: ‘C.1.01.01’ (‘actor’,...), ‘C.1.01.02’ (‘object’,...) and ‘C.1.01.03’ (‘VAT law’,...).

Let us now turn to the example of ‘transaction for which no VAT is required’ and see how this UoU, including its subcategories, fits in the overall structure of the categorisation framework. In figure 4 we included the top node of figure 2 as well as the third UoU which was directly associated with the top node, i.e. ‘VAT law’ (‘C.1.01.03’). If the UoU of ‘transaction for which no VAT is required’ appears at granularity level 3, the framework is structured as follows:

⁸ The level of depth depends on the level of granularity that one allows in the overall structure of the framework [9]. For instance, if no further specification of the notion of ‘exemption’ was required in the VAT regulatory ontology, all the terms used to denote the different subcategories would be assigned the same label.

**Table 2.** Example of term annotations

Term	Annotation
vrijstelling	<Nl-Be><C.4.01030101.01>vrijstelling</C.4.01030101.01></Nl-Be> <Nl-Be><C.4.01030101.02>vrijstelling</C.4.01030101.01></Nl-Be> <Nl-Be><C.4.01030101.03>vrijstelling</C.4.01030101.02></Nl-Be>
niet onderworpen aan BTW	<Nl-Be><C.4.01030101.03>niet onderworpen aan BTW</C.4.01030101.03></Nl-Be> <Nl-Be><C.4.01030101.04>niet onderworpen aan BTW</C.4.01030101.04></Nl-Be>
exemption	<Fr-Be><C.4.01030101.01>exemption</C.4.01030101.01></Fr-Be> <Fr-Be><C.4.01030101.02>exemption</C.4.01030101.02></Fr-Be> <En-UK><C.4.01030101.01>exemption</C.4.01030101.01></En-UK> <En-Ir><C.4.01030101.03>exemption</C.4.01030101.03></En-Ir>
zero-rated	<En-UK><C.4.01030101.02>zero-rated</C.4.01030101.02></En-UK> <En-Ir><C.4.01030101.03>zero-rated</C.4.01030101.03></En-Ir>
outside the scope of VAT	<En-Ir><C.4.01030101.03>outside the scope of VAT</C.4.01030101.03></En-Ir>

Whenever a termontographer maps a term to the categorisation framework, the term will receive a categorial label. A label which provides information about the language and cultural setting of each term (see above) is added as well. Table 2 shows, for each term in table 1, the possible annotation(s).

In the next section we will discuss some of the implications this method has with respect to the difficulties of representing multilingual information.

4.2 Representing Multilingual Information

Working with the categorisation framework has a number of important implications for the compilation of a multilingual termontological database and consequently for multilingual ontology engineering.

1. The search for more textual material will be restricted to only those texts which deal with categories that are present in the framework.
2. The decisions one should make during the termontography search phase (see section 2) with respect to the relevance of particular terms also rely upon what is already present in the framework (and how much granularity is allowed). Only terms referring to UoUs in the framework should be extracted, labelled and structured (either manually or automatically).
3. The process of aligning the meaning-related terms (including (near-)synonyms)⁹ from the different languages via a non-culture specific categorisation framework¹⁰ is improved in comparison with the EuroWordNet approach (section 3). Moreover, the framework allows us to verify whether relevant terms are missed out.
4. Due to the fact that intercategory relationships specify how UoUs are related to one another, the mapping from terms to the enriched framework might reveal differences in the way particular UoUs are perceived throughout the different legislations.
5. Assigning the UoU and relationship identifiers (see section 4.1) to particular linguistic patterns in the domain-specific texts will result in a semantically-annotated corpus which can be used, in a later stage, to automate processes like knowledge extraction, knowledge classification or multilingual alignment of terminology found in multilingual domain-specific corpora. Semantically annotated corpora may also be used for a wide variety of NLP applications such as document classification, text summarisation and information retrieval.

5 Conclusion

In this paper we reflected on the complexity one encounters when trying to represent multilingual information about the European VAT regulatory domain in a termontological database which could be used as a possible resource for multilingual refinement within the DOGMA ontology engineering framework. We proposed the termontography methodology, and showed by means of a UoU framework how to overcome the difficulties that arise when aligning and representing multilingual information in a termontological database. These difficulties were due to differences in language and culture.

⁹ A synonymy relation between two terms occurs if these terms have exactly the same termontological specification. This is in the case of a term like ‘value added tax’ and its abbreviated form ‘VAT’. Two words are near-synonyms if they essentially refer to the same category while having slight meaning differences. An example of near-synonymy are words like ‘fraudster’ and ‘swindler’.

¹⁰ The same idea, but applied to general language, is presented in Villegas et al. [21].

As the termontography method is in an early stage of development a number of issues still need further elaboration. To name but a few:

- How to account for possible meaning variations between lexicalisations (in different languages) referring to the same category?
- Is it possible to speed up the process of multilingual terminology alignment by making use of both the categorisation framework and external resources like electronic versions of multilingual dictionaries, glossaries or term lists?
- How to guarantee the reusability of the categorisation framework?

Although being in an early stage of development, we believe that the termontography method will prove to be a valuable contribution to the development of a VAT regulatory ontology. Which and how many UoUs need to be specified by field specialists depends on the VAT regulatory application the ontology is going to be used for.

Acknowledgments. This research is performed within the FFPOIROT project (IST 2001-38248). This project aims at compiling for several languages (Dutch, Italian, French and English) a computationally tractable and sharable knowledge repository for the financial forensics domain. We would like to thank Patrick Wille and Isabelle Desmeyere of VAT Applications NV (<http://www.vatat.com/www/en/vatapp.htm>) for providing us with examples from the VAT regulatory domain.

References

1. Temmerman, R. 2000. *Towards New Ways of Terminology Description. The sociocognitive approach*. Amsterdam: John Benjamins.
2. Temmerman, R. 2001. "Sociocognitive Terminology Theory". In: M. T. Cabré & J. Felui *Terminología y cognición*. Barcelona: Pompeu Fabra.
3. Steve, G., A. Gangemi & D.M. Pisanelli. 1997. Integrating Medical Terminologies with ONIONS Methodology. Technical Report ITBM-RIM-3/97. Istituto Tecnologie Biomediche, CNR.
4. Gangemi, A., D.M. Pisanelli & G. Steve. 1998. "Ontology Integration: Experiences with Medical Terminologies". In: N. Guarino (ed.) *Proceedings of the Conference: Formal Ontology in Information Systems*. Italy. IOS Press.
5. Gangemi, A., D.M. Pisanelli & G. Steve. 1998. "Some Requirements and Experiences in Engineering Terminological Ontologies over the WWW". *Proceedings of the 1998 Knowledge Acquisition Workshop*. Canada.
6. Kerremans, K., R. Temmerman & J. Tummers. forthcoming. "Termontography: a units-of-understanding approach".
7. Jarrar, M. & R. Meersman. 2002. "Formal Ontology Engineering in the DOGMA Approach". *1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE '02)*, *Lecture Notes in Computer Science*, Vol. 2519, 1238–1254. Berlin: Springer-Verlag.
8. Gómez-Pérez, A., M. Fernandez & A. De Vicente. 1996. "Towards a Method to Conceptualize Domain Ontologies". *Workshop on Ontological Engineering. ECAI'96*, 41–51. <http://delicias.dia.fi.upm.es/miembros/ASUN/ECAI96.ps>.

9. Fernandez, M., A. Góméz-Pérez & N. Juristo. 1997. "METHONTOLOGY: From Ontological Art Towards Ontological Engineering Workshop on Ontological Engineering". *Spring Symposium Series. AAAI97*, Stanford, USA.
<http://delicias.dia.fi.upm.es/miembros/ASUN/SSS97.ps>.
10. Sure, Y. & R. Studer. 2003. "A methodology for Ontology-based Knowledge Management". In: John Davies, Dieter Fensel & Frank Van Hamelen (eds.) *Towards the Semantic Web. Ontology-Driven Knowledge Management*, 33-46. New York: John Wiley & Sons
11. Sager, J.C. 1990. *A practical course in terminology processing*. Amsterdam: John Benjamins.
12. Temmerman, R., F. Simonis & L. Luyten. 1990. *Terminologie: een methode*. Leuven: Acco.
13. Cabré, M. T. 1999. *Terminology. Theory, methods and application*. Amsterdam: John Benjamins.
14. Agnesund, M. 1997. "Representing culture-specific knowledge in a multilingual ontology – an object-oriented approach". *Proceedings of the workshop on ontologies and multilingual NLP*, Nagayo, Japan. <http://citeseer.nj.nec.com/agnesund97representing.html>.
15. Bryan, M. 2002. "MULECO – Multilingual Upper Level Electronic Commerce Ontology". CWA Version 4.0.
<http://www.ecimf.org/events/Brussels-20020220/MULECO-CWA-02-02.doc>.
16. Vossen, P. 1998. *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*. Dordrecht: Kluwer Academic Publishers.
17. Pianta, E., L. Bentivogli & C. Girardi. 2002. "MultiWordNet: Developing an aligned multilingual database". *Proceedings of the First International Conference on Global WordNet*, Mysore, India.
18. Lauser, B. T. Wildemann, A. Poulos, F. Fisseha, J. Keizer & S. Katz. 2002. "A Comprehensive Framework for Building Multilingual Domain Ontologies: Creating a Prototype Biosecurity Ontology". *Proceedings of the International Conference on Dublin Core and Metadata for e-Communities*, 113-123. Firenze: University Press.
<http://www.bncf.net/dc2002/program/ft/paper13.pdf>.
19. Halpin, T. 2001. *Information Modelling and Relational Databases. From Conceptual Analysis to Logical Design*. San Francisco: Morgan Kauffman.
20. Vouros, G. & E. Eumeridou. 2002. "Simple and EuroWordNet. Towards the Prometheus ontological framework". *Terminology* 8:2, 245-281.
21. Villegas, M., N. Bel, A. Lenci, N. Calzori, N. Rummy, A. Zampolli, T. Saduri & J. Soler. 2000. "Multilingual linguistic resources: from monolingual lexicons to bilingual interrelated lexicons." *LREC 2000 2nd International Conference on Language Resources & Evaluation*. <http://www.ub.es/gilcub/lascosas/pubYreps/parolesimple.pdf>.

Ontology-Based Platform for Trusted Regulatory Compliance Services

Henry Ryan¹, Peter Spyns², Pieter De Leenheer², and Richard Leary³

¹ Lios Geal Consultants Ltd, Galway Technology Centre, Ireland
henryryan@eircom.net

² Vrije Universiteit Brussel – STAR Lab, Pleinlaan 2, Building G-10,
B-1050 Brussel, Belgium

{Peter.Spyns, Pieter.De.Leenheer}@vub.ac.be
www.starlab.vub.ac.be

³ Advanced Forensic Solutions Ltd, Staffordshire, UK
rleary@af-solutions.co.uk
www.af-solutions.co.uk

Abstract. This paper describes the shared objectives and planned collaborative venture to develop and deploy an ontology platform for trusted regulatory assurance and compliance certification services. The platform is based on extensions to the established DOGMA suite using legal domain-experts to develop and deliver the use cases. It is end-to-end, based on national and EU regulations, and will exploit multi-lingual representations, commitment rules, ontology management and reconciliation, intelligent agents and forensic tools. Evolution of the platform to a common integrated workbench for data privacy and digital rights controllers and for generic regulatory use is outlined.

1 Introduction

This paper originates from the collaborative plan¹ prepared by European and Canadian experts to integrate ontology research within the legal/regulatory world. The intent is to comprehensively manage data privacy and digital rights across jurisdictional and language barriers, using state of the art ontology and risk management tools, in systems to be used by non-computer experts. The multi-disciplinary participating teams integrate extensive practical and academic expertise in many domains including ontology technology, natural language processing and data mining, systems engineering and database management, regulatory compliance, and in the interpretation and application of European and national laws and regulations governing data privacy and digital rights management.

The objective of this paper is to disseminate early information and to engage the wider ontology and regulatory compliance community in the establishment of a proven workbench for the specific application areas and its extension to more generic regulatory use. The background to business case for regulatory ontologies and the

¹ Based on concepts developed by Luk Vervenne, VUB Ontology Competence Centre and Steve Kenny, independent Data Privacy Technology Consultant.

rationale underlying the DOGMA platform developments are presented first. A description and assessment of the state of the art and associated activities in the six main challenge areas is then outlined. The paper concludes with a series of invitations to support and challenge this work.

2 Industry Issues in Regulatory Compliance

Businesses and government must be able to show compliance of their outputs, and often also of their systems and processes, to specific regulations. Demonstrable evidence of this compliance is increasingly an auditable consideration and required in many instances to meet acceptable criteria for good corporate governance. Moreover the number and the complexity of applicable regulations in Europe and elsewhere is increasing. This includes mandatory compliance audits and assessments against numerous regulations and best practice guidelines over many disciplines and against many specific criteria. The implementation of information communications technology also means that previous manual business processes are now being performed electronically and the degree of compliance to applicable regulations depends on how the systems have been designed, implemented and maintained. Keeping up with the rate of new regulations for a major corporation and small business alike is a non-ending task. What is the answer to all of this regulatory complexity? First one should simplify regulations where possible and then apply automatic tools to assist.

The automated data demands of networked economies and an increasingly holistic view on regulatory issues are driving and yet partially frustrating attempts to simplify regulations and statutes. In an ideal world companies and other organizations would have the tools and online services to check and measure their regulatory compliance; and governmental organizations would be able to electronically monitor the results. This requires a more systemic shared approach to regulatory assurance assessment and compliance certification.

Lessing [37] has a simple yet profound thesis "Code is law". The application of this concept taken in conjunction with the emergence of "regulatory ontologies" opens up a new way of assessing whether burgeoning systems are compliant with regulations they seek and claim to embody. First specific regulations (e.g., data privacy, digital rights management) are converted into and expressed as "Regulatory Ontologies". These ontologies are then used as the base platform for a "Trusted Regulatory Compliance Certification Service". In other words the ontology approach starts by representing existing regulations in specific domains as ontologies and then proves compliance and internal consistency by assessing and validating policy documents and systems to the analysed regulations. Where discrepancies arise in the ontological representation of regulations applicable to the same domain of interest the approach provides the capabilities to automatically drill down and isolate the relevant data (texts, inferences and processes).

Over time the resulting ontology describing and managing the areas analysed can literally replace the regulations + compliance criteria. So much so, it is envisaged that an eventual outcome could be that the formal writing (codification) of future laws in areas covered by the paper (and eventually those in other areas also) will start with the derived ontologies and use intelligent agents to help propose specific legal text which

ensures that the policy objectives are correctly coded in law. In addition automatic generation of networked computer applications that are perfectly compliant with the wide variety of directives and laws in any country is one of the ultimate goals of this type of ontology based work.

3 Business Case for Legal Ontologies

There are two prevalent definitions of formal ontologies: [21] defines an ontology as an *explicit, formal specification of a shared conceptualization of a shared domain* whereas [23] defines an ontology as a logical theory accounting for the intended meaning of a formal vocabulary.

The goal of the current work is to ‘validate’ applications conformance with existing regulations using derived multi-lingual regulatory ontologies. If an application can “commit” to such an ontology it is basically compliant to the regulations. The initial use cases selected for development and application are data privacy and digital rights management. The business and social rationale for these selections is considered to be clear. For many multinational organizations, EU Directive 95/46/EC is becoming the “de facto” global data protection standard. In addition the roll out of broadband and the practically insatiable demand for multimedia content makes it increasingly necessary to manage digital rights effectively subject to the established concepts of authorship and intellectual property of published works.

Because of their richer semantic content and organization than former approaches, such as thesauri, ontologies bring the opportunity to effectively manage and widely disseminate structured domain knowledge. Following the current trend for development of “application ontologies” each organization interprets the respective legal and regulatory frameworks and composes light-weight “application level ontologies” for their own specific purposes. The danger here is the total lack of any common basis for building these application ontologies, leading to disparate vendor specific conceptualizations and multiplicity, lack of standardization and variable levels of protection. In the case of data privacy for instance, this diversity of semantics/ontologies would make it increasingly difficult to reflect the sought for compliance or accurately provide ‘data protection’ as described by Directive 95/46/EC.

The two regulatory domains chosen aim at providing businesses the opportunity to integrate regulatory compliance into their systems and/or to be better advised on their compliance needs. Once this knowledge is modeled into an ontology it can be shared by all users in each specific regulatory domain. As a result, websites and information systems can achieve an improved, coherent, more transparent and more readily auditable regulatory compliance level, which will ultimately cost less to implement and maintain. The regulatory governing bodies will also be able to make use of this innovative technology (i.e. ontology-based regulatory compliance services), to improve their audits, compliance checking, etc. However, a note of caution is required. Technology alone will not provide the answer.

The administration of law must be supported by the community being administered and policed. Also, absolute compliancy across EU to all regulations is an impossible

goal simply on the basis of the complexity and logistical issues involved. Inevitably there will be organizations who, though non-compliant, acted with good intentions but for example were simply unable to comply due to logistical problems. This work therefore envisages a significant focus particularly at the deployment stage on *Self Declaration of (non)Compliance* and accreditations each assisted by the regulatory ontology platform technology to help achieve self imposed recognition, acceptance and fulfillment of responsibilities. This concept of self declaration for compliance already applies and works satisfactorily in facets of tax, environment and insurance law.

4 Use of DOGMA

DOGMA² is a research initiative of VUB³ STARLab where various theories, methods and tools for extracting representing, storing, browsing and using formal ontologies are being studied and developed. The current DOGMA infrastructure for ontological engineering comprises an ontology server and a modeling tool.

The DOGMA approach adopts the classical database model-theoretic view in which the ontology is decomposed into a lexon base, consisting of (multiple) plausible ‘intuitive’ conceptualization(s) of a domain, and a layer of ontological commitments representing domain rules, constraints, etc. Each conceptualization is simplified to a set of context-specific binary fact types, called lexons, formally described as $\langle \gamma \text{ term1 role corole term2} \rangle$, where γ is a context identifier used here to group lexons that are logically related to each other in the conceptualization of the domain. For instance $\langle \text{investment: person offers/offered by prospectus} \rangle$ is a lexon that may hold for some application, expressing in this case that within the context of investment, a *person* (term1) may plausibly have *prospectus* (term2 or tail) occur in an associated role *offers* (with *offered by* as co-role).

This separation between facts (lexons) and rules is called the principle of *double articulation*. It enables two essential capabilities: shared meaning for concepts and relationships; and the degrees of freedom necessary to enable any application to have its own specific interpretation (i.e. ontological commitment) expressed in the form of constraints and rules in terms of that shared lexon base. Each such ontological commitment defines a *partial semantic account of an intended conceptualization* [22].

Concept types are assumed to stand for the things themselves independent of any lexical representation. Within the formalization (such as a computer representation) these concepts are represented by specific lexical terms within a certain context. A context is represented by an abstract context identifier $\gamma \in \Gamma$, where Γ is the context space which informally refers to a source (e.g., an extract from a legal text, the information in that legal text, the thing the information is about). A context identifier defines therefore a disambiguating mapping to bind terms to concepts, relations or instances. For example the lexical term (or English word) “interest” has several senses, meanings, depending on the section of text, non-linguistic situation, or

² Abbreviation for Developing Ontology-Guided Mediation of Agents

³ Vrije Universiteit Brussel, Brussels, Belgium

context. In the “banking” context, the term “interest” refers to a fixed fee for borrowing money. If it is used in the “investment” context it refers to a share or stock. For more details on formal contexts in DOGMA see [10].

In a concrete sense the DOGMA approach takes “agreed semantic knowledge” out of an IT application in much the same way as databases take data structures out of applications. Thus within DOGMA ontologies in general achieve a form of semantics independence for IT applications that may be compared to the earlier notion of data independence for such applications using databases [42].

The underlying technology used to store the ontology, the related linguistic terms and its instances is an RDBMS. This database approach to infrastructure is being used to ensure *robustness and scalability* [32] for the targeted regulatory ontologies. Performance scalability can be controlled by DOGMA’s inherent properties as the DOGMA ontology server will provide for ontologies the services a regular relational DBMS provides for databases: controlled access to and efficient definition, creation, maintenance etc., of these ontologies. The DOGMA approach and architecture, by virtue of its “model-theoretic database” style, is well suited for building complex and large-scale applications, in the same way that the model-theoretic view of databases enables scalable technology and successful high-performance DBMS, tools and applications industry. For a more complete description of DOGMA see [33,49].

5 The Challenges

The main challenges addressed in the research methodology are summarized below in Table 1. This covers the six phases - repositories, access tools, ontology development, integrated access, analysis tools, compliance checking as outlined in [35]. Here these features are grouped differently to better reflect the activities associated with layering and the DOGMA engineering developments.

5.1 Secure Ontology Framework

Security, auditability and trust are issues of *meta-knowledge* which transcend individual domains and involve regulatory knowledge. As ontologies in general evolve into ever more accurate resources of knowledge about a given domain (e.g., local legislation), applications will increasingly become dependent on a *trusted* (and even implicitly *assumed*) quality of these resources. This in turn implies that the design, engineering, and maintenance of ontologies (both their lexons –terminology relationships– and the allowed commitments –rules, constraints, procedures) must become part of an auditable and therefore secure “compile time” process, supported by tools, and possibly controlled by an accepted authority (including secure and trusted communication).

On the other hand, relevant ontologies may serve as standard reference models for such –even multilingual– legal texts. Compliance with a particular law or regulation then becomes *defined* as the formal (auditable) “commitment” to its underlying ontology, verifiable by software agents consulting it, and/or compliance can be *generated* from such an ontology, becoming a part of a software system that for

Table 1. Major activities and description

Major Activities	Description
Secure Ontology Framework	Create a framework for secure and trusted collection, storage, evaluation, certification and distribution of ontology-based regulatory knowledge
Generic semantic portal infrastructure	Establish a semantically-based multilingual portal infrastructure that includes dialogue technology for use by domain experts and application developers, and that offers access to regulatory ontology navigation services
Engineering / development infrastructure and tools	Extend the STAR Lab DOGMA to a scalable and robust ontology engineering infrastructure with integrated ontology management tools, an integrated collaborative ontology development environment, web services and agent infrastructure
Regulatory ontology content	Use and adapt formal methodologies for terminological and ontological engineering, for setting-up ontology-driven regulatory compliance, for defining regulatory ontology usage by means of semantic web services, agents and rule systems, for creating semantic UI, for making existing software systems “compliance-aware” by means of an ontology (agent technology, mediation)
Layered models for the selected use cases	Model the multiple ontology layers involved in a full fledged regulatory framework and set up interoperable ontology-based upper application ontologies to be used by the uses cases and related Privacy Enhancing Technologies/Digital Rights Management applications
Evaluation and deployment	Utilize and evaluate the generic applications of the infrastructure, tools, services, methodologies and ontologised content

instance requires a "validated compliance *with specific regulation stamp*". Research on and development of such reference models is expected to establish a more "scientific" approach to standardization. Therefore, applications and the manner in which they commit to lexons also will need to be governed "at runtime" by secure processes, possibly to be generated from domain ontologies and from relevant meta-knowledge.

5.2 Generic Semantic Portal Infrastructure

Currently few semantic portals exist – e.g., OntoWeb [45], SEAL [39], OLR [11]. Almost all are research prototypes containing limited amounts of data or are not applied on a large scale. A specific query mechanism with a dedicated conceptual query language will also be adopted, based on the experiences of the OntoWeb semantic portal [48]. Such ontology servers have already been developed – e.g., Sesame based on RDF [4], but without a well developed contextual view on multi-

linguality and terminology since each application domain has its special jargon [53]. Synonyms and translations become problematic in that view.

The portal represents the “visual side” of an ontology-based search engine. Users can browse a concept tree and/or type some keywords. In the current practice, a user types in human language terms that the search engine tries to match with the metadata annotations. Obviously several languages have to be taken into account to allow the users to use the ontology in their native language for information.

In general, it is believed that multi-lingual language terms i.e. terms in several Human Languages such as regulatory terms - are linked to a concept (e.g., uniquely labeled by a specific head-term or concept label). By doing so, the conceptualization achieves a high degree of language independence [54] in keeping with the definition of an ontology as “a shared and agreed conceptualization” [56].

An extra layer is needed to transform the linguistic input provided to the portal by a user to an input expressed by conceptual labels. Dialogue technology – see e.g., [60] – will be used to perform this task with the added value that the user will no longer be restrained to specific keywords. In principle, complete sentences can be entered. Initial experiments have indicated that ontology-based searching yields better and faster results [31]. However it is recognized that the human-computer interaction (HCI) will change with the advent of dialogue technology combined with genuine semantic processing.

All these capabilities will be used to develop a next generation regulatory expert system. Using interactive interviews this will provide correct ontology-based and rules driven regulatory advice either for end-users or for legal advisors seeking support. In this way expert knowledge can be ontologized [26] and provided in stages to the knowledge base, and its usage will no longer be governed / limited by the availability (presence or absence) of the expert person [8]. The use of a *regulatory ontology* also enables the retrieval of relevant documents without consideration of the source language.

5.3 Engineering/Development Infrastructure and Tools

Not many “serious” ontology servers (e.g., OntoLingua [14] or see [12] for an overview) and related extensive tool suites (see [13] for an overview) are available nowadays. In particular few are successfully focused on regulatory compliance. The most “popular” ontology tools are editors (e.g., OntoEdit [51], WEBODE [7]), annotators (e.g., OntoMat-Annotizer [29]) and alignment and merge components – e.g., Prompt [44] or Chimaera [40]. However, when critically reviewing the tools, it becomes clear that many of them are research prototypes and have a limited functionality. On the other hand, many papers provide descriptions on what specific tools should achieve (e.g., version control for ontologies [36]). Protégé [44] from Stanford seems a very widely used environment.

STARLab is developing a comprehensive architecture and matching software tool suite (called the DOGMA Studio™) for ontology engineering, alignment, storage, manipulation, maintenance, and deployment in general. At present this suite provides basic modeling functionality. Further implementation work on the DOGMA Studio™ is ongoing and the software system is being extended, and/or deployed according to state of the art software engineering principles and standards. Extensions to the

DOGMA Studio™ from a design scalability point of view will be applied in the first place in the DOGMA Modeler engineering tool.

The domain experts working on conceptualizing and modeling the ontologies will - in most cases - be specialized lawyers. It is very unlikely that these legal experts will be accustomed to model with ORM. More convenient would be to allow these people to model in a natural language input way, perhaps aided by an interactive CASE-like dialogue system. For example, if inserting of invalid lexons or concepts, or adding rules inconsistent with the lexon base is detected, online guidelines can be presented.

Ontologies are not static objects. During its design and life-cycle, a regulatory ontology will grow and shrink over time or domain, as a result from "ontologizing" lots of legal data. Furthermore, collaborative modeling of overlapping aspects within one module, is a concurrent process and hence there is a requirement for design-scalable modeling. This can be met by a future concurrent versioning mechanism wrapped around the DOGMA server, adopted to deal with concurrency control (as in DBMS) and other changes that may cause consistency problems for the applications that commit to the ontology. The versioning system will serve as the backbone for a distributed modeling environment, interfaced by DOGMA modeler.

Modeling of ontologies in general is a task where multiple disciplines come together and "agree". Presently the emphasis is more on design scalability than on performance scalability. Ontology experts, domain experts and legal ontologists *work collaboratively in a distributed environment* on respectively different or overlapping aspects of the ontology being engineered, which are expected to be large in the data privacy and rights domain. Two intertwined aspects of design scalability predominate: a scalable organization and deployment of a large number of ontological commitments and a provision of an ontology modeling environment for engineering these commitments collaboratively in a distributed environment.

Ontologies used for regulatory compliance services require extensive mining of large volumes of fairly expertly structured (and multilingual) documents, to be interpreted and aligned according to "hand-crafted" upper ontologies (e.g., for *data privacy* and *digital rights*). A major goal is to significantly advance the state of the art in (especially shallow) NL parsing technology, term extraction, and concept analysis, and in the technology to support the abstraction and alignment mechanisms that are required. The automatic lexon extraction tools will use information extraction (IE) technology [1,5] such as that developed using the GATE system (<http://gate.ac.uk/> [2,6]) and that developed by Expert System (Italy). The selected extraction technology will use ontologies and reasoning, combined with language analysis components (e.g., parser, morphological analyser) to identify automatically relations. These techniques have been applied successfully to other domains, e.g., MUMIS [47] - automated multimedia metadata extraction and indexing from football videos and text, SOCIS [46] - IE and image processing in the domain of police crime scenes, EMPATHIE [30] - Enzyme and Metabolic Path Information Extraction.

A fundamental premise is that other people can learn how to apply and (re-)use the results. Currently existing research methodologies and industry practices cover specific parts of semantic web area e.g., [15,52] for ontology development; [23] for ontology consistency checking and OntoClean [24] for taxonomy refinement. A formal database modeling method has therefore been adopted and will be used to develop a formal ontology engineering methodology (independent of ontology

implementation languages), with existing formal methods, including ONIONS⁴. The regulatory ontology *modeling methodology* will be based on Object Role Modeling (ORM) [25]. ORM is a well known and widespread database formal modeling methodology, and very suited for ontology modeling. Applying ontology modeling techniques that are close to familiar methods will ease the experts' knowledge engineering process.

Regulatory ontology modeling requires more commitment specification flexibility than presently is possible in ORM. Therefore STARLab is researching and developing a formal commitment specification language called Ω -RIDL (Ontology Reference and IDea Language), which is based on the earlier syntactical principles of RIDL [41], a conceptual query, update and specification language for databases. The expressive power of ORM is a subset of the expressive power of Ω -RIDL. Ω -RIDL may further be extended for ontology querying, and also made compatible with the new DAML ontology query language as part of the platform development.

5.4 Regulatory Ontology Content

Regulations such as data privacy and digital rights management for broadcasting are typically based upon European level directives providing the principles, but the legal implementable regulation is situated at the national level. Furthermore legal arguments are drawn from supportive legislation. This whole "regulatory framework" is to be taken into account.

Currently, the number of usable regulatory ontologies is almost non-existent. One notable although small example of a formal ontology for regulatory compliance is the work carried out in the industrial research project "IKF-IF-LEX", (cf. [18]) in the domain of anti-money-laundering regulations. At the same time there is quite a lot of "legal ontology" material available: van Kralingen and Visser functional ontology of Law [57], Breuker's ontology [3], Stamper's "NORMA" [50], Tiscornia's Jur-IWN [55], Gangemi's DOLCE+Jur [18], (cf. [61]). There is also a considerable body of pre-ontology material e.g., terminologies such as EuroWordNet [59]. Other foundation work on which it is intended to build include the results of Working Group on Legal Ontologies from within the SIG on Content Standards of the OntoWeb EC Project, and the DOLCE foundational ontology developed as a module of the Foundational Ontology Library within the EC Project WonderWeb (<http://wonderweb.semanticweb.org>) [62]. The latter is being already used in the DOLCE+Jur core ontology of Law.

⁴ There are the ONIONS and ONIONS-II [17,18,20] methodologies for ontology creation and merging that have been successfully applied to several domains (bio-medical, legal, fishery). They are independent from a particular formal language, but both assume first-order logic, some classification service, and a foundational ontology as a unifying paradigm. Although they are not yet completely engineered, they contain nearly complete guidelines on how to start from scratch and/or to reuse existing sources, in order to build domain ontologies.

5.5 Layered Models for the Selected Use Cases

A first requirement for modeling ontologies that support regulatory compliance applications in the domains of data privacy and rights management implies structuring of ontologies into *layers* (horizontal decomposition, reuse-oriented) and *modules* (vertical decomposition, task-oriented) that are related. The ontology model has to cope with large amounts of concepts and relations to extract and abstract, and to support interpretation of the texts relative to different contexts.

The layering methodology makes it possible for different modelers to collaboratively develop ontology modules in a distributed environment, model different aspects of the domain (general to specific, or vertical), and finally integrate and merge these modules. Afterwards these modules can also be reused for other tasks in different application ontologies. Modularization and layering enables smooth efficient ontology maintenance, allowing several regulations (that don't necessarily agree) to co-exist in the same ontology, preserve alternative axiomatizations for the same concept and maximize the reusability of the ontology among different *kinds* of applications. In DOGMA terms each ontology module may be seen as an instance of an "ontological commitment". Conversely, ontological commitments may be seen as reusable components of knowledge.

An application ontology can consist of many layers, but at least the following four are present in most ontology libraries, and will be present in the envisaged ontologies as well.

- *Foundational layer*, containing domain-independent concepts and relations (e.g., part, participant, object, regulation, organism, activity). The *WonderWeb* Project ontology library will be reused: it is being developed by integrating and partly aligning different foundational proposals, for example the DOLCE ontology [<http://wonderweb.semanticweb.org>] that has already been applied to the legal domain.
- *Core layer*, containing concepts and relations (e.g., provision, article, obligation, legal person) that are central and generic for the domains being considered. Core modules are defined by using elements from foundational modules. Some proposals for core legal ontologies exist, but they still need to be harmonised (some work has been carried out by a dedicated Working Group in the *OntoWeb* Project [<http://ontology.ip.rm.cnr.it/legontoweb.html>]).
- *Domain layer*, containing concept and relations for the domains (e.g., consent, confidentiality, user, traffic data). Domain modules are defined by using elements from core modules
- *Application layer*, containing concepts and relation (e.g., package, encoding) typical of the particular application for which the library is maintained. Application modules are defined by using elements from domain modules

5.6 Evaluation and Deployment

Where available, general ontology surveys and papers on evaluation are descriptive in nature. Some methodologies have been presented – e.g., OntoClean [24] – to check and evaluate an ontology (or more precisely a hierarchy) on its consistency. However,

implemented tools based on these methodologies are rare and rather immature – e.g., [52]. Certain empirical evaluations have been carried out in order to assess the effectiveness of the state-of-the-art in ontology management tools [28], but to ensure constancy, further appraisals need to be done, especially across diverse sets of domains.

The proposed approach guarantees that the project results actually address real business needs and will have an impact on business practices. The impact will be first assessed in Europe, Canada and other regions that have strong relevance from a business perspective, such as the Accession countries, and the US. At present Australia does not have a data protection regime deemed to offer adequate protection. They could offer an interesting case for a future application and analysis. This is to be investigated as part of a wider international roll out of the business models and strategy.

6 Next Steps

This paper presents a *novel approach to regulatory compliance*. For successful deployment the results will be integrated into a coherent and professional education and training programme in order to achieve awareness, knowledge dissemination and market acceptance. Once the ontology engineering infrastructure, tools and methodology are well in place and have proven their validity, they can be applied to many other regulatory compliance application domains. New partner organisations can then be influenced by the new regulatory ontology domain to put proper compliance procedures in place either by

- engaging in compliance integration or for their systems/applications
- setting up advisory expert systems
- using the *regulatory ontology platform* as a basis to build very specific application ontologies.

The core partners committed to implementing the concepts of this paper are essentially complete. However the partners are seeking interest from additional external user/tester partners who would undertake to apply the approach to new applications within the initial two domains and to initiate applications in other regulatory domains. These new partners will have access to the methodologies and software components developed and will undertake in return to test the validity and generic applicability. This development cycle and evaluation process will be based on content as provided by the new consortium members.

It is intended to issue a specific call for interested parties. In the meantime inputs on any of the following issues is sought

- Do you have contact suggestions for interested people/fields?
- What are examples of how you could use such a system?
- What would make the system more useful?
- How will such a system be deployed?
- How are the problems addressed currently dealt with?
- What are some existing technologies that should be investigated?
- What are recommendations for other issues to be addressed?
- What complementary tools should be developed?

Acknowledgements. The authors gratefully acknowledge the anonymous reviewers and the following expert colleagues for their generous assistance and contributions to the ideas contained in or underlying this paper: Andre Biesheuvel, Kalina Bontcheva, Hamish Cunningham, Francesco Danza, Jaime Delgado, Jan Demey, Jan Dhont, Aldo Gangemi, Meinhard Gans, Mike Gurski, Mustafa Jarrar, Adil Hameed, Stephen Kenny, Larry Korba, Johannes Kraus, Silvia Llorente, Heike Lohmar, Sophie Louveaux, Caterina Lupo, Diana Maynard, Robert Meersman, Stefan Poslad, Stephan Schneider, Rita Temmerman, Ruben Verlinden, Luk Vervenne, Richard Wright, Gang Zhao.

Peter Spyns is supported by the IWT (OntoBasis project GBOU 2001 #10069), and Richard Leary by the EU (project FF POIROT IST #2001-38248).

References

1. Appelt D., (1999), An Introduction to Information Extraction. *Artificial Intelligence Communications*, 12(3):161–172
2. Bontcheva K., Kiryakov A., Cunningham H., Popov B., & Dimitrov M., (2003), Semantic Web Enabled, Open Source Language Technology. *Language Technology and the Semantic Web*, Workshop on NLP and XML (NLPXML-2003), held in conjunction with EACL 2003
3. Breuker J., Winkels R., & Valente A., (1997), A core ontology for law. In van Marcke K. & Daelemans W. (eds.), *Proceedings of the 9th NAIC*
4. Broekstra J., Kampman A., & van Harmelen F., (2002), Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, Springer-Verlag LNCS 2342, pp. 54–68
5. Cunningham H., (1999), *Information Extraction: a User Guide (revised version)*. Research Memorandum CS-99-07, Department of Computer Science, University of Sheffield
6. Cunningham H., (2002), GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, Volume 36: 223–254
7. Corcho O., Fernández-López M. Gómez-Pérez A., (2002), Evaluation experiment for the editor of the WebODE ontology workbench, in Sure Y., & Angele J., (eds.), (2002), *Proc. of the First International Workshop on Evaluation of Ontology based Tools (EON 2002)*, vol. 62 of *CEUR Workshop Proceedings* (<http://CEUR-WS.org/Vol-62/>), p.122–134
8. Correa da Silva, F.S., Vasconcelos W.W., Robertson D.S., Brilhante V., de Melo A.C.V., Finger M. and Agustí J. On the insufficiency of ontologies: problems in knowledge sharing and alternative solutions, *Knowledge-Based Systems*, 15 (3) March 2002: 147–167
9. Cunningham H., (1999), *Information Extraction: a User Guide (revised version)*. Research Memorandum CS-99-07, Department of Computer Science, University of Sheffield
10. De Bo, J., Spyns, P. (2003), Extending the DOGMA framework in view of multilingual ontology integration. Technical Report 09, STAR Lab, Brussel.
11. Dhraief H., Nejdil W., Wolf B., & Wolpers M., (2001), Open Learning Repositories and Metadata Modeling, in *Proc. of the first Internat. Semantic Web Working Symposium (SWSW01)*, pp. 495–514
12. Ding Y. & Fensel D., (2001), Ontology Library Systems: the key to succesful Ontology Re-use, in Horrocks I. & Hendler J., *Proc. of the first international Semantic Web Conference (ISWC 2002)*, LNCS 2342, Springer, pp. 93–112
13. Duineveld A, Stoter R., Weiden M., Kenepa B. & Benjamins R., (2000), Wondertools? A comparative study of ontological engineering tools, in *International Journal of Human-Computer Studies* 6(52):1111–1133.

14. Farquhar A., Fikes R., Pratt W. & Price J., (1995), Collaborative Ontology construction for Information Integration, Knowledge Systems Laboratory Stanford University,-Technical Report 95-63
15. Fernández-López M., Gómez-Pérez A., Euzenat J., Gangemi A., Kalfoglou Y., Pisanelli D., Schorlemmer M., Steve G., Stojanovic L., Stumme G., & Sure Y., (2002), A survey on methodologies for developing, maintaining, integrating, evaluating and reengineering ontologies, *OntoWeb deliverable #D1.4*, Madrid
16. Gangemi, A., Pisanelli, D.M. & Steve, G., (1999), An Overview of the ONIONS project: Applying Ontologies to the Integration of Medical Terminologies, in *Data and Knowledge Engineering*, vol.31
17. Gangemi A., Pisanelli D.M., Steve G., (2000), Understanding Systematic Conceptual Structures in Polysemous Medical Terms, in *Proceedings of the 2000 AMIA Fall Symposium*
18. Gangemi, A., Pisanelli D., & Steve G., (2001), A Formal Ontology Framework to Represent Norm Dynamics, in *2nd International Workshop on Legal Ontologies*, Univ. of Amsterdam
19. Gangemi A., Guarino N., Masolo C., & Oltramari, A., (2003), Restructuring WordNet's Top-Level To appear in *AI Magazine*
20. Gómez-Pérez A., Angele J., Fernández-López M., Christophides V., Stutt A., Sure Y., et al., (2002), A survey on ontology tools, *OntoWeb deliverable #D1.3*, Madrid
21. Gruber T., (1993), A translation approach to portable ontology specifications, in *Knowledge Acquisition*, volume 5, number 2, pp. 199-220
22. Guarino N. & Giarretta P., (1995), Ontologies and Knowledge Bases: Towards a Terminological Clarification, in *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, N. Mars (ed.), IOS Press, Amsterdam, pp. 25-32
23. Guarino N., (1998), Formal Ontologies and Information Systems, in Guarino N. (ed.), *Proceedings of FOIS98*, IOS Press, pp. 3-15
24. Guarino N. & Welty C., (2002), Evaluating Ontological Decisions with OntoClean, in *Communications of the ACM*, 45 (2): 61-65.
25. Halpin T., (2001), *Information Modeling and Relational Databases: from conceptual analysis to logical design*, Morgan-Kaufmann, San Francisco.
26. Hameed A., Sleeman D. & Preece A., (2002), Detecting Mismatches Among Experts' Ontologies Acquired through Knowledge Elicitation, *R&D in Intelligent Systems XVIII: Proc. of ES2001 – the 21st SGES International Conference on Knowledge Based Systems & Applied Artificial Intelligence*, pp. 9-24, Cambridge, U.K., Dec 2001, Springer, London. http://www.csd.abdn.ac.uk/~ahameed/hameed_es2001_kbs2002.pdf
27. Hameed A. (2002). *Ontology-enabled Knowledge Sharing & ReUse* (PhD proposal) www.csd.abdn.ac.uk/~ahameed
28. Hameed A., Preece A. & Sleeman D., (2003), Ontology Reconciliation, in *Handbook on Ontologies in Information Systems*, Staab S. & Studer R. (eds.), LNCS, Springer Verlag
29. Handschuh S., Staab S. & Ciravegna F., (2002), S-CREAM – Semi-automatic CREAtion of Metadata, in Gómez-Pérez A. & Benjamins R., (eds.), (2002), *Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web (EKAW 2002)*, LNAI 2473, Springer, pp. 565-570
30. Humphreys K., Demetriou G., & Gaizauskas R., (2000), Two Applications of Information Extraction to Biological Science Journal Articles: Enzyme Interactions and Protein Structures. *Proc. of the Pacific Symposium on Biocomputing (PSB-2000)*, Hawaii, pp. 505-516.
31. Iosif V. & Mika P., (2002), EnerSearch virtual organisation case study: Evaluation document, *On-to-Knowledge deliverable #D29*, Malmö, Sweden
32. Jarrar M. & Meersman R., (2002), Scalability and Reusable in Ontology Modeling, *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine (SSGRR2002s)*, [only available on CD-ROM]

33. Jarrar M. & Meersman R., (2002), Formal Ontology Engineering in the DOGMA Approach, in Meersman R., Tari Z. et al., (eds.), *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings*, LNCS 2519, Springer Verlag, pp. 1238–1254
34. Jarrar M, Demey J., & Meersman R., (2003), On Reusing Conceptual Data Modeling for Ontology Engineering, In Aberer K., Spaccapietra S., & March S., (eds), *Journal of Semantics of Data Vol. 1*, Springer, (2003), 1 (1): [in print]
35. Kerrigan S., Lau G., Zhou L., Wiederhold G., & Law K., (2001), Information infrastructure for regulation management and compliance checking, in *National Conference on Digital Government Research*, Los Angeles CA , pp. 167–170
36. Klein M. & Fensel D., (2001), Ontology versioning on the Semantic Web, in Cruz I., Decker S., Euzenat J. & McGuinness D. (eds.), *Proceedings of the First Semantic Web Working Symposium (SWWS 2001)*, pp. 75–91
37. Lessing L., (1999), *Code: and other laws of Cyberspace*. Basic Books
38. Lindberg D, Humphreys B, & McCray AT, (1993), The Unified Medical Language System. *Methods of Information in Medicine* 32 (4): 281–91.
39. Maedche A., Staab S., Stojanovic N., Studer R. & Sure Y., (2003), SEmantic PORTal – the SEAL approach, in Fensel D., Hendler J., Liebermann H. & Wahlster W., (eds.), *Spinning the Semantic Web*, chapter 11, pp. 461–518
40. McGuinness D., Fikes R., Rice J. & Wilder S., (2000), an Environment for Merging and Testing Large Ontologies, in Cohn A., Guinchiglia F. & Selman B., (eds.), *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Morgan Kaufmann, San Francisco, pp. 483–493
41. Meersman, R., (1982), The high-level end user, in *Data Base: The 2nd generation*, Infotech state of the art report (vol. 10, no. 7), Pergamonn Press, U.K.
42. Meersman R., (2001), Ontologies and Databases: More than a Fleeting Resemblance, in d'Atri A. and Missikoff M. (eds), *OES/SEO 2001 Rome Workshop*, Luiss Publications
43. Noy N., Ferguson R., & Musen M., (2000), The knowledge model of Protégé-2000: Combining interoperability and flexibility, in *Knowledge Engineering and Knowledge Management. Methods, Models, and Tools*, 12th Internat. Conf. EKAW 2000, LNCS 767, Springer
44. Noy N. & Musen M., (2002), promptdiff: A fixed-point algorithm for comparing ontology versions, in *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI2002)* [also available as SMI Technical Report SMI2002-0927]
45. Oberle D. & Spyns P., (2003), OntoWeb – Knowledge Portal, in Staab S. & Studer R., (eds.), *Handbook on Ontologies in Information Systems*, LNCS, Springer, pp. 521–540
46. Pastra K., Saggion H., Wilks Y., (2003), Intelligent Indexing of Crime-Scene Photographs, *IEEE Intelligent Systems*, Special Issue on "Advances In Natural Language Processing", 18(1): 55–61.
47. Saggion H., Cunningham H., Bontcheva K., Maynard D., Hamza O., Wilks Y., *Multimedia Indexing through Multi-source and Multi-language Information Extraction: The MUMIS Project. Data Knowledge Engineering Journal*. 2003.
48. Spyns P., Oberle D., Volz R., Zheng J., Jarrar M., Sure Y., Studer R., Meersman R., (2002), OntoWeb - a Semantic Web Community Portal, in Karagiannis D., & Reimer U. (eds.), *Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM02)*, LNA 2569, Springer Verlag, pp. 189–200
49. Spyns P., Meersman R. & Jarrar M., (2002), Data modelling versus Ontology engineering, in Sheth A. & Meersman R. (ed.), *SIGMOD Record Special Issue* 31 (4), pp. 12–17
50. Stamper R., (1980), LEGOL: Modelling legal rules by computer. In Niblett B., (ed.), *Computer Science and Law*, pages 45–71. Cambridge University Press

51. Sure Y., Erdmann M., Angele J., Staab S., Studer R., & Wenke D., (2002), *OntoEdit: Collaborative ontology development for the semantic web*, in Horrocks I. & Hendler J., *Proc. of the first international Semantic Web Conference (ISWC 2002)*, LNCS 2342, Springer, pp. 221–235
52. Sure Y., (2003), *Methodology, Tools & Case Studies for Ontology based Knowledge Management*, Karlsruhe (unpublished PhD thesis), AIFB Karlsruhe
53. Temmerman R., (2000), *Towards New Ways of Terminology Description. The Sociocognitive Approach*. Amsterdam/Philadelphia: John Benjamins.
54. Temmerman R., (2003), *Innovative methods in Specialised Lexicography*, in *Terminology [to appear]*
55. Tiscornia D., (2001), *Ontology-driven Access to Legal Information*, DEXA 2001-Workshop LISA, Legal Information Systems and Application Munich
56. Ushold M. & Tate A., (1998), *Putting ontologies to use*, in *The Knowledge Engineering Review*
57. van Kralingen R., Visser P., Bench-Capon T., van den Herik H., (1999), *A principled approach to developing legal knowledge systems*. *International Journal of Human Computer Studies* 51(6): 1127–1154
58. Verheyen G. & van Bekkum P., (1982), *NIAM, aN Information Analysis Method*”. In: *IFIP Conference on Comparative Review of Information Systems Methodologies*, T.W. Olle, H. Sol, A. Verrijn-Stuart (eds), Noord-Holland
59. Vossen, P. (eds), (1998), *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*, Kluwer Academic Publishers, Dordrecht
60. Webb N., De Roeck A., Kruschwitz U., Scott P., Steel S., & Turner R., (2000), *Evaluating a Natural Language Dialogue System: Results and Experiences*, in *Proceedings of the Workshop on From Spoken Dialogue to Full Natural Interactive Dialogue: Theory, Empirical Analysis and Evaluation (at the 2nd International Conference on Language Resources and Evaluation LREC2000)*
61. <http://www.ontoweb.org/download/deliverables/D3.4.pdf>
62. <http://wonderweb.semanticweb.org/deliverables/documents/D15.pdf>

Business Contract Obligation Monitoring through Use of Multi Tier Contract Ontology

Vandana Kabilan, Paul Johannesson, and Dickson M. Rugaimukamu

Department Of Computer and System Sciences
Royal Institute of Technology and Stockholm University, SE 164 40, Sweden
{vandana,pajo,si-dmr}@dsv.su.se

Abstract. A core business enterprise function is trading with other business organizations. Such trade relationships are governed by legal contracts. The obligations and fulfillment conditions as stipulated in a business contract demand performance of business activities in the pre-defined choreography. Non-performance could lead to serious legal and/or financial implications. Therefore, it is essential to have integrated business and contract workflow management. Domain knowledge of legal business contracts can be represented using the proposed *multi-tier contract ontology*. This paper analyzes and models the different *obligation types* found in a contract and also identifies *obligation states* through which each *obligation* passes through. Thereafter, the paper proposes a methodology to monitor contractual obligation through the use of Multi tier Contract Ontology and related contract fulfillment process modeled as *Contract Workflow Model*.

1 Introduction

Legal business contracts are established to agree upon the rules and regulations to govern the business transactions in between the agreement partners. A contract also stipulates and testifies to the legal binding nature of the agreement, as well as the remedial options and rights granted to the partners in case of any disagreement or dispute. Information contained in a contract needs to be assimilated as knowledge, which can be analyzed and reused in the business process management efficiently.

We find various research approaches related to e-commerce or business process modeling or contracting. Most contract management tools available today view the contract as a document [35], some as a work flow process [1], while others have represented the contract terms in deontic logic [2] or other AI methodologies [3,4]. Legal dictionaries are also being developed [36]. But any proposed methodology should take in to account the implications of all the domains, which influence a contract, namely:

- Business Domain
- Legal, Regulatory Domain
- Technological and Information Systems Domain.

A successful contract is negotiated such that the terms are beneficial to all the parties involved. Thus, a business entity should aim to execute the contract

relationship in an *optimized flow* so as to gain the *maximum efficiency* and *performance compliance* and in turn *maximum profit*. The contract stipulates both implied and explicit *obligations*, which are *legally binding*, and need to be *fulfilled* by the execution of *performance as agreed*. Therefore, a contract once negotiated and signed cannot be forgotten. It needs to *govern and monitor* the actual execution of the business processes. Similarly, existing business or workflow models influence the negotiation and acceptance of the contract terms and obligations. A business entity should not accept such terms and conditions, which he is unable to comply or fulfill. Existing methodologies fail to bridge the chasm in between the two domains.

Contract compliance and optimum business process management can be achieved if the existing gap between the two areas is bridged. Contract obligation monitoring and tracking is one of the crucial issues to be resolved in this context. This paper proposes a methodology to monitor and track the contract with respect to the business process execution. In order to do so, the paper makes use of the Multi-Tier Contract Ontology [45]. The paper focuses mainly on the analysis of different types of obligations, stipulated in a contract and the various states through which each obligation passes through. Based on the multi tier contract ontology conceptual models, specifically focusing on the *obligations* and their related *fulfilling performances*, a Contract Workflow Model (CWM) is deduced.

The rest of this paper is organized as follows: In Section 2, we present works of other researchers related to our field of interest and have influenced our research methodologies. Next, we present an overview of the entire Multi-Tier Contract ontology in Section 3. In Section 4 we focus on conceptual models for the Upper Core Layer of the Multi-Tier Contract Ontology followed by a detailed analysis of contract *obligation types* and *obligation states* in Section 5. Detailed conceptual models for the domain and template layers maybe referred to in [48]. In Section 6, the use of Contract Workflow Model for obligation monitoring is illustrated. Detailed discussion on the methodology for Contract Workflow Model deduction is out of the scope of this current paper. Finally, the paper concludes with a discussion on other possible applications and potential impact of Contract Knowledge and future work (Section 7).

2 Related Research

This research is guided by the ontology design concepts and guidelines as suggested by T. Gruber [5], who proposes *clarity*, *coherence*, *extensibility*, *minimal coding bias* and *ontological commitment* as fundamental objectives for any ontology. This paper also adopts the philosophy of Noy and McGuinness [6] that ‘an ontology defines a common vocabulary for researchers who need to share information in a domain’. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. This work has been inspired by similar works on ontology of Guarino [7], who has worked on formalizing ontological principles to knowledge engineering methodologies. The use of UML as an ontology-modeling tool has been discussed by some researchers, including Cranefield [46], Gerd Wagner [47].

Increasing use of ontological engineering in the realm of business process engineering has been supported by Deborah McGuinness [8] who identifies the goals for ontological development and proposes guidelines for a centralized knowledge base to capture semantics from domain, standard terms and vocabularies. Howard

Smith states [9], “Agent based system rely upon ontologies to provide a map or blue print of the space each agent navigates through communication and interoperation with other agents” and he also emphasizes on the need for software engineering, business process re engineering and ontological engineering integration.

Different E-Contracting approaches such as event driven action semantics [10], component based view as in COSMOS [11] or as Meta schema for workflow [1] exist.

Though the workflow aspect has been the focus in [1], the analysis of the contract has been done from a document centric perspective. Daskalopulu, in [12,13] analyses and represents obligation and business exchange transition states through deontic logic. An e-market controller as suggested in [12] implies strict adherence and compliance to the contract specifications. Milosevic [14] proposes the use of Subjective Logic for a Discretionary Enforcement Moderator agent for automated dispute resolution after contract violations. Milosevic has also expressed contractual obligations using genetic software engineering behavior trees in [15]. He uses deontic logic to translate the behavior trees to business process flows. Our approach is based on similar premises as of Milosevic, however, our approach is an integral perspective of contract knowledge and business process workflows. Moreover our representation using conceptual models and workflow patterns to model the contract workflow is easily comprehensible, coherent, and extensible. W.M.P. van der Aalst [16] has proposed several workflow patterns for modeling business workflow, as have also E Sivaraman and M Kamath [17]. Ronald Lee, one of the pioneers of e contracting, has used Petri nets to model documentary credits procedures in [33].

Most of the above work pertains to the realm of e contracting. Some focus of electronic contract negotiation and signing, some on electronic monitoring and enforcement agents, others on technical communication standards like the ebXML initiative, EDI etc. Given the nature of contractual obligations and their executions, a rigid automated monitoring as proposed by Milosevic or Daskalopulu is not practical. As most business strategy and decision making is still carried out by human counterparts, it becomes crucial for the decision makers to have a clear understanding of the contractual obligations and possible repercussions in case of contract violations. Thus, this paper proposes a methodology for semi automated monitoring of contracts based on fostering a clear, comprehensible understanding of contractual obligations, and the performance required to fulfill them. In this regard, the methodology is applicable to contracts established electronically or traditionally across the table.

3 Multi Tier Contract Ontology

Contracts can be of diverse types [25]. The domain and scope of each type varies largely. Thus a single contract ontology layer would be too large as well as diverse in nature to be of any practical use. In order to capture the entire range of business related contracts in a single framework; we propose a layered, multi tier contract ontology, which moves from the abstract to the specific Meta data definition progressively down the stratifications. Currently, we have identified a minimum of three different layers as presented below. Other extensions and layers may be possible.

- The **Upper Core Layer** represents a general composition of a contract, which may be applicable across most of the prevalent types of contracts. The concepts defined here may be considered to be the atomic blocks based on which all other contract types may be defined. Fundamental concepts like *role*, *consideration*, and *obligation* are defined here, as presented in section 4.
- The second layer is **Specific Domain layer** or contract type specific collection of several contract type ontology. Each ontology represents a specific contract type like property lease rental, employment contract, and sale of goods amongst others. Each contract type inherits all fundamental features of the upper layer and thereafter specializes on the specific knowledge particular to that contract domain. The focus of our research has been on business and corporate contract types. Initially, we have concentrated on sale and purchase of goods contract domain. In this context, we have based our conceptual models on the UNCISG [37], UNIDROIT [40] principles for International Commercial Contracts [39] and UNCITRAL [38] model law and recommendations. We have also mapped to other existing ontologies like product catalogues (UNSPSC [22], CPV [23]), payment terms (standard payment methods), and delivery terms (ex INCOTERMS [41]), which can be applicable across diverse contract types. Each contract type is an extension or restriction of the upper layer contract ontology to a specified domain. The obligations, rights and their implications have been analyzed based on Jan Ramberg's book on International Commercial Transactions [24].
- The third layer, **Template Layer**, consists of a collection of template like definitions for established or recommended contract models like the International Chamber of Commerce's [18] contract model for International Sale of Goods [19], European Union's SIMAP [20] online procurement contract models etc. The template layer is visualized as a detailed definition of a particular contract type and could be a recommended contract form based on a contract type as modeled in the shared domain layer. The template layer may bind the conceptual components to specific range of values, thereby limiting their re-usability outside their domain of application.

The Multi Tier Contract Ontology models explicit, declarative and strategic knowledge contained within a contract. Interdependencies and relationships between the components of a contract are represented. The three-tier contract ontology is a stratified presentation of information, moving from the most generic *global contract ontology* domain to the *shared contract ontology* domain and further down to implementation specific *template contract ontology*.

The *procedural knowledge* and *strategic knowledge* as presented through the choreography of obligations and performance events in the contract, is then modeled as a set of Contract Workflow Models (CWM)(Section 6). The actual business workflow should be as close and similar to the deduced CWM to be compliant with the contract conditions.

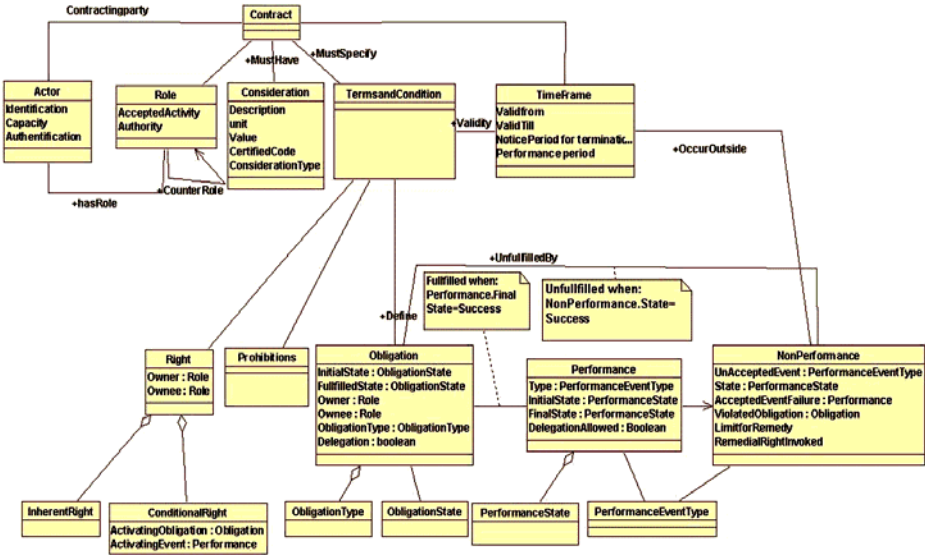


Fig. 1. Outline for Upper Level Contract Ontology

4 Upper Core Layer Contract Ontology

Figure 1 is an extract from the current working model for the global upper layer. We illustrate the concepts through the following business case scenario.

Assuming a hypothetical case scenario (Fig. 2) where a seller promises to sell cars to another person (buyer) in return for some amount of money. A *promise* is a statement of commitment to fulfill some act or perform certain deeds. When a promise is made with the legal intent to back it up in any court, it becomes a *legal obligation* or *commitment*.

Cars are the object or *consideration* of the Promise. Legally, a consideration is any material or abstract benefit or thing of value for the exchange of which two parties agree to enter in to a contract. It could also be simply a promise to fix a leaky roof or a promise not to do something [21,28]. ‘*Selling*’ becomes the *performance*, which *fulfills* the above promise. The promise to sell is an *obligation*; which is realized only when the actual business act of giving the cars in return for the money is performed. An *obligation* must have a *Receiver* or an *Acknowledger* or a *Claimant* or a *Beneficiary* of the obligation, that is one who receives the consideration or is the recipient for whom the promised act is being performed or carried out. The Buyer who buys the cars and pays money for it is the *Receiver* of the obligation to sell.

Contracts are legal testament to the promises (Obligations/commitments) made by the two parties to each other and spell out the details constituting the obligations, the accepted boundaries and performance criteria like time, venue etc. The two parties *promise* to carry out certain acts in order to *fulfill* their commitments, this presumption can be held valid if and only if the two parties have the necessary

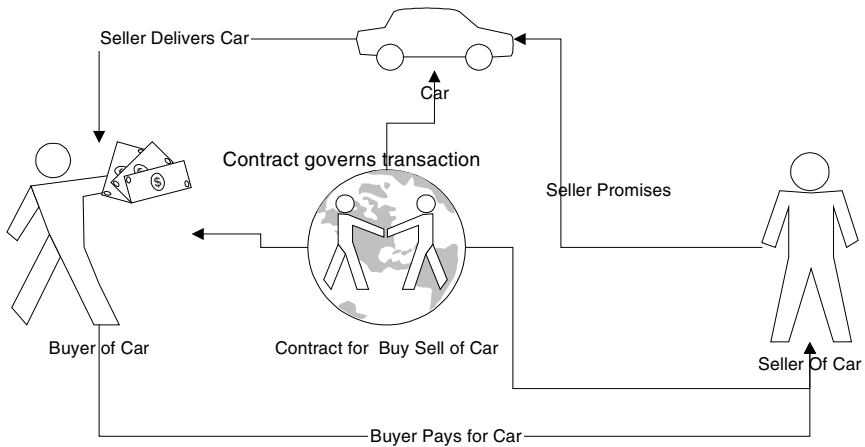


Fig. 2. Business Case Scenario Example

capability or *competence* or *ability to perform* the said deeds. That is the seller promising to sell the car must *own* the car and the buyer must have the money to pay for it. Lastly the contract must outline the *terms and conditions* under which the promised performance shall occur. In a normal situation, a contract is executed as planned and agreed. In case the expected performance does not occur within the expected *time frame* or occurs in an unsatisfactory manner or is delayed due to forces beyond the control of the promisor, then the primary obligation goes in to a state of un-fulfillment. The occurrence of the *Non Performance* event activates certain pre-agreed *rights* on behalf of the *promisee*. In this case, the buyer may have the right to seek remedy in the form of some interest or penalty, or may choose to terminate in accordance with the agreement. But the *buyer* may also choose not to do anything and could settle the issue by mutual agreement. However, whatever course of action the *buyer* chooses, the *seller* (the promisor of the primary obligation) is bound to a *reconciliatory promise* to fulfill the remedy asked. This secondary or reconciliatory promise may also include the fulfillment of the initial primary promise to sell.

The above was a very simple case scenario where we see that obligations have different natures; obligations may give rise to other obligations and rights. Rights too may give rise to new obligations being formed etc. In the following section, we present the obligation type and obligation state model extracts from the Upper Core Level Contract ontology.

5 Obligation Types and Obligation States

As proposed by others like Ronald Lee [33], Yao-Hua Tan [10] and Daskalopulu [3,4], we acknowledge that the statements in a legal contract are informative, declarative or performative. Informative statements pertain to the description of the actors, the subject of the contract, jurisdiction and other pertinent information. Declarative or performative statements are statements of intentions or conditions,

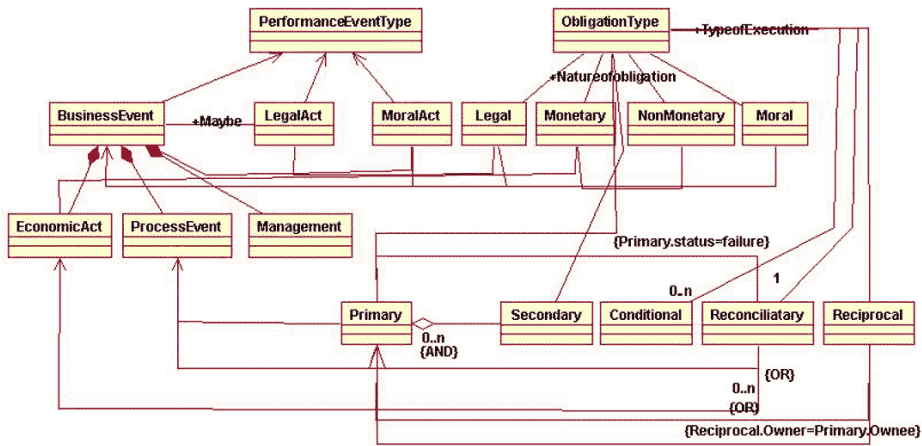


Fig. 3. Common Obligation Types

which change states through the performance of specified conditions. They are usually of the following types:

- 1. **Obligations:** Obligations are mandatory. They have an obligation owner, who is the recipient of the obligation and there is an obligation ownee, who is the one who has to carry out the obligation. It is mandatory for the ownee to execute the obligation condition once and only once in every single execution of the contract.
- 2. **Permissions/Rights:** Permissions or Rights too have a rights Owner and a ownee, but the *right* is executed by the owner in this case. The execution of a right is of an optional nature or it may be conditional depending on the execution of some other obligation.
- 3. **Prohibitions:** Prohibitions are statements of whatever actions should *not* be executed or whatever actions may be unacceptable to either or both parties.

Obligations are bound to their performative actions or non-performative actions for their fulfillment. Based on the nature of their fulfillment execution, we propose a categorization of obligations in to the following abstract categories, for ease of grouping and understanding, refer Fig. 3 above:

- 1. **Primary obligation:** has to be fulfilled whenever the contract is to be executed and is the principal objective behind the agreement itself. Like the obligation of a seller to deliver goods in conformity to the contract or that of a buyer to accept and pay for goods as ordered.
- 2. **Reciprocal obligation:** could be a primary obligation in itself, but is also the obligation expected to be fulfilled by the counter party in response to the execution of the primary obligation. E.g. the buyer's obligation to pay is reciprocal to the seller's obligation to deliver and vice versa. Also the buyer's obligation to pay is a primary obligation for the buyer.
- 3. **Conditional obligation:** does not need to be activated under the normal course of events. Most remedial rights and obligations come in this category, like the

buyer's right to seek compensation for failed delivery, binds the seller to deliver the goods at his added cost etc.

4. **Secondary obligation:** gets activated as a part of some other obligation. Like the seller's obligation to package goods suitable for transportation could be said to be sub unit of his primary obligation to deliver.

Also based on the contextual nature of the obligation, which require the specific type of performance, business, legal or ethical as the case maybe, obligations can also be grouped as:

- **Legal Obligation:** In a legal business contract all statements are legally binding and have legal implications. However, we propose the category of legal obligation to differentiate those obligations that require some specific legal actions to be carried out to fulfill them. The legal acts may or may not be part of the business management process
- **Business Obligation:** Similarly, business obligations are also legally binding. But we propose this term to categorise all those obligations, which are specifically related to business performance. Business obligations are further typified as *monetary obligation* and *non-monetary obligation*. Monetary obligations are those that deal with some economic transactions or exchange. We summarize that all business obligations need not necessarily be economic commitments. Commitments like the buyer has to send the order once he has contracted to purchase goods or that the buyer has the obligation to arrange for carrier and inform the seller etc require business execution but do not have economic implications in between the buyer and the seller. We term these types of obligations as *non-monetary obligations*.
- **Moral/Ethical Obligation:** Yet another category is moral or ethical obligations. These may not be severely binding but are more practically or socially expected obligations. Like for example, though it maybe the obligation of the buyer to pick up the goods from the seller's premises, he may request the seller to help in arranging the transportation or require some other help in arranging the official customs duty etc. The seller, though he may not be legally bound to assist the buyer, he is morally bound to aid the buyer subject to the buyer requesting such help.

Individual obligations may belong to more than one category simultaneously. In order to track the contract fulfillment process more systematically, we have identified some stages in the contract execution phases wherein each contract obligation may be said to go through certain 'Obligation States'. The proposed obligation states are similar to most common business activity states and are also adapted from similar obligation state definitions proposed by Lee [33], Yao-Hua Tan [10] (Fig. 4).

Figure 5 illustrates an example of obligation state changes in response to the business performance events for a typical *obligation to deliver* on behalf of the seller.

- **Inactive:** Every obligation can be said to be in 'inactive' state once the contract has been signed but contract execution is yet to be started. The obligation will remain in 'inactive' state in between cycles of contract execution also.
- **Active:** An obligation may be said to be 'active' when the triggering performance event has been issued. For example, once the buyer sends the Purchase Order to the seller and he receives it, then the seller's obligation to deliver is triggered.

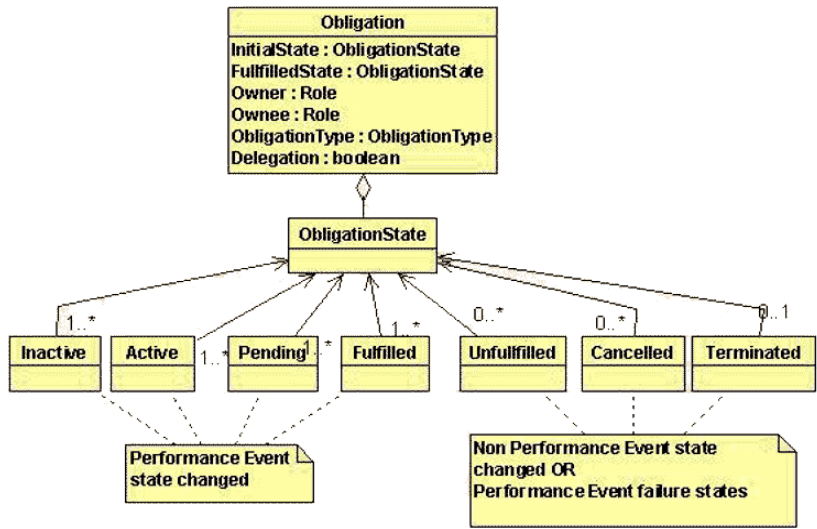


Fig. 4. Obligation State Models

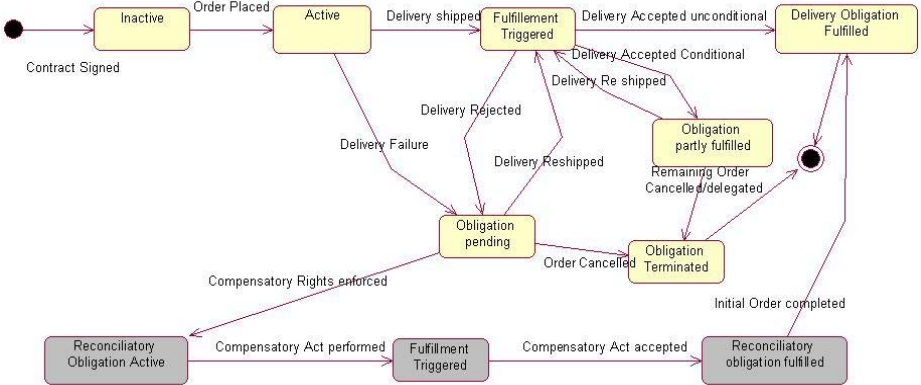


Fig. 5. State Transformation Diagram of Obligation

- **Pending:** When fulfillment activity from performer’s side has been accomplished but the acceptance from the other party is still awaited. When the seller has dispatched the goods from his warehouse, and is waiting for the third party carrier to deliver the goods to the buyer. Alternatively, an obligation may remain in the pending state, till the all the necessary fulfillment criteria are satisfied. The buyer may have received the goods but may have rejected the goods as being unsatisfactory.
- **Fulfilled:** when the performance conditions are satisfied within the stipulated performance conditions. Once the buyer inspects the goods and accepts the delivery.

- **Terminated/Cancelled:** if the activation of the obligation is terminated or cancelled by the Obligee, mutually, or legal authority.

6 Contract Workflow Model

In this section, we discuss one possible application of the proposed contract knowledge base. Multi tier Contract Ontology provides information regarding the obligations and their expected performance activities. From the contract obligation models (Figs. 3, 4) we get the states through which an obligation can pass through. Combined together, we get procedural knowledge regarding the expected *flow of business actions* within individual contracting parties' business process workflow. This expected choreography of business actions is modeled as a *Contract Workflow Model* using workflow patterns and notations.

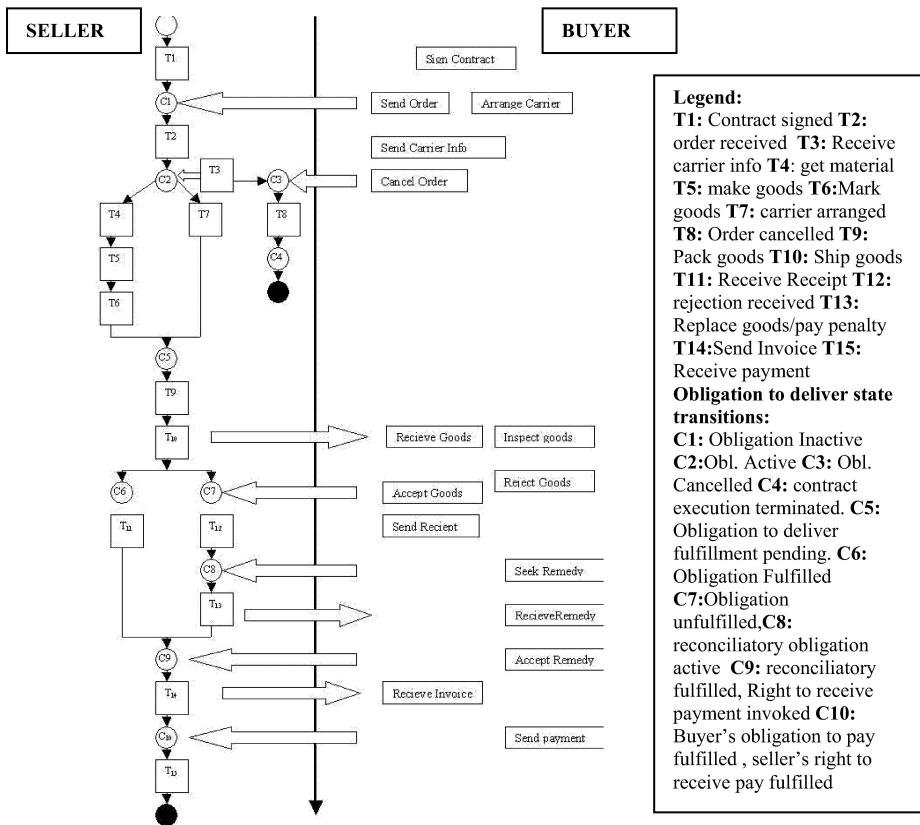


Fig. 6. Sample CWM for a seller's obligations in a sale of goods contract.

The Contract Workflow Model (CWM) is deduced from the instance data and informative knowledge present in the contract instance and the conceptual models for

the same contract type (from Multi tier Contract Ontology). It is the expected workflow pattern, which complies with the agreed upon obligation and fulfillment pattern. The Contract Workflow Model would then be the reference for comparison with existing internal business process or shared workflow models of the business partners. Ideally, the existing business process flow should be identical with the contract workflow pattern or at least within tolerable limits of performance and contract compliance.

The Contract Workflow Model is also instrumental in enabling automated or semi-automated tracking of obligation fulfillment. The Contract workflow model identifies individual obligations and the various states they go through in relation to the occurrence of related business or legal events. Thus, the obligation state transition can be tracked and monitored with respect to the actual occurrence of the identified business event in the business process workflow of each individual party. A mapping to business process workflow or existing business management systems can be made based on the identified obligation state transition and business activity interaction point. For the current phase of human-to-human knowledge transfer a guideline for deducing the CWM has been designed. But, a semi-automated deduction logic using Object Constraint Language for UML is the subject of ongoing work.

7 Conclusion

In this paper we have identified the existing gap between business process management and business contract execution and fulfillment. The paper has also identified the common obligation types that are present in a business contract and has also proposed an obligation state identification for denoting the stages through which every contract obligation passes.

We have also illustrated the use of Multi tier Contract Ontology for deducing the contract workflow model as a means to affect contract and business process integration as well as to monitor the contract fulfillment process.

Contracts and Business are established and carried out by human agents at some level or the other. Contracts are testament to agreed intentions of human partners and their actual behavior cannot be predicted. Thus, it is impractical to have rigid automated agents for contract enforcement or for negotiation. However, an easy accessible knowledge pool can provide invaluable aid in the decision-making and process management regarding contractual obligations and their required fulfillment. Semi-automated decision support or obligation tracking agent tools are within the scope of our ongoing research work.

The main benefit of having a centralized contract knowledge base is that a business organization may effectively increase profitability by reusing knowledge gained from legal domain, from their previous experience. Future work in this field includes analysis and modeling of other types of business and legal contracts. Migration to human to machine, machine to human methodologies are possible extensions to ongoing work.

References

1. Kamalakara Karlapalem, Ajay R Dani and PP. Radha Krishna; A frame Work for Modeling Electronic Contracts; ER 2001, LNCS 2224 pp. 193–207
2. Yao-Hua Tan, Walter Thoen. Using Event Semantics for Modeling Contracts. Proceedings of 35th Hawaii International Conference on System Sciences–2002.
3. A.Daskalopulu. Logic Based Tools for Legal Contract Drafting: Prospects and Problems. Proceedings of the 1st Logic Symposium 1997. University of Cyprus Press, pp. 213–222
4. A Daskalopulu & MJ Sergot (1997). The Representation of Legal Contracts, AI and Society 11(Nos. ½), pp. 6–17
5. T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, March 1993.
6. Ontology Development 101: A Guide to Creating Your First Ontology by Noy, N.F. and McGuinness
7. Guarino, N. 1992. Concepts, Attributes and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases. *Data & Knowledge Engineering*, **8**: 249–261.
8. Conceptual Modeling for Distributed Ontology Environments, Deborah L McGuinness, published online at www.ontology.org also, in the proceedings of the Eight International Conference on Conceptual Structures Logical, Linguistic, and computational issues (ICCS 2000).
9. The role of ontological engineering in B2B Net Markets, Howard Smith, director of strategy, e business, CSC Europe, www.ontology.org
10. Yao-Hua Tan, Modeling Directed Obligations and permission in Trade Contracts. 31st Annual Hawaii International Conference on System Sciences, vol 5, 1998.
11. Griffel, M. Boger, H. Weinreich, W. Lamersdorf, M. Merz. Electronic Contracting with COSMOS – How to Establish, Negotiate and Execute Electronic Contracts on the Internet. EDOC '98, 1998
12. Assapassia Daskalopulu, Evidence Based Electronic Contract Performance Monitoring. *The INFORMS Journal of Group Decision and Negotiation*. Special Issue on Formal Modeling in E-Commerce, 2002.
13. Daskalopulu A. & Maibaum T. S. E. (2001). Towards Electronic Contract Performance. *Legal Information Systems Applications*, 12th International Conference and Workshop on Database and Expert Systems Applications, IEEE C. S. Press, pp. 771
14. Zoran Milosevic, Audun Jøsang, Mary Anne Patton, Theo dimitrakos ,Discretionary enforcement of Electronic Contracts, EDOC 2002
15. Z.Milosevic, R G Dromey. On expressing and Monitoring Behaviour in Contracts, 6th International Enterprise Distributed Object Computing Conference (EDOC'02), 2002.
16. W.M.P. van der Aalst. The application of PetriNets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998
17. E Sivaraman, K Kamath, On the use of Petri nets for business process modeling', *Proceeding of the 11th Annual Industrial Engineering Research Conference*, Orlando, FL., May 2002
18. International Chamber of Commerce, <http://www.iccwbo.org/>
19. ICC International contract for sale of goods, published by ICC books, 2002
20. European Union SIMAP , <http://simap.eu.int/>
21. Online Law dictionary, <http://dictionary.law.com/>
22. United Nations Standard Products and Service Codes (www.unspsc.org)
23. Common Procurement Vocabulary (CPV) <http://simap.eu.int/EN/pub/src/main5.htm>
24. Jan Ramberg; ICC Guide to Incoterms 2000. Understanding and Practical Use; International Chamber of Commerce 2000
25. Contract types, www.findlaw.com
26. Resource Description Framework, www.w3.org

27. J. Hendler and D. L. McGuinness. (2000, November). The DARPA Agent Markup Language. *IEEE Intelligent Systems*. [Online]. 15 (6), pp. 67-73.
28. Duhaime, Lloyd, Duhaime's law dictionary, www.duhaime.org.
29. Suresh R, Knowledge Management –an overview, available online at http://www.kmadvantage.com/docs/km_articles/km_an_overview.pdf (accessed on 4th June 2003)
30. Protégé 2000, ontology editor tool, available at <http://protege.stanford.edu/>, last accessed on 5th June 2003
31. Unified Modeling Language, <http://www.uml.org/>, accessed on 5th June 2003
32. Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web," *Scientific American*, May, 2001
33. Ronald M Lee, facilitating International Contracting: AI Extensions to EDI, published in *International Information Systems*, January 1992
34. David S. Linthicum, Leveraging Ontologies & application Integration, published on *EAI Journal*, June 2003.
35. Metalex , XML standard for mark up of legal resources , <http://www.metalex.nl/> ,last accessed on 18th June 2003
36. Legal RDF Dictionary, European Legal RDF Dictionary initiated by John Mc Clure, <http://www.lexml.de/rdf.htm>
37. United Nations Convention on Contracts for the International Sale of Goods, 1980, available online at (accessed on 12th June 2003)
<http://www.jus.uio.no/lm/un.contracts.international.sale.of.goods.convention.1980/doc.html>
38. United Nations Commission on International Trade And Law. <http://www.uncitral.org/>, accessed on 12th June 2003
38. UNIDROIT principles of International Commercial Contract, 1994, available online at <http://www.unidroit.org/english/principles/pr-main.htm>, last accessed on 12th June 2003
39. International Institute for Unification of Private Law, <http://www.unidroit.org/english/conventions/c-main.htm>
41. INCOTERMS 2000, can be got from ICC, http://www.iccwbo.org/index_incoterms.asp
42. Enterprise Ontology, developed by the Enterprise Project, Edinburgh, <http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html>
43. Business Object Reference Ontology,
44. REA enterprise Ontology, <http://www.msu.edu/user/mccarth4/rea-ontology/>
45. V Kabilan, P Johannesson, D Rugaimukammu, An ontological approach to Unified Contract Management, to be published in the proceedings of 13th European Japanese Conference on Information Modeling and Knowledge Bases , held on June 6–7th 2003, Kitakyushu, Japan
46. Cranefield, S., and Purvis, M. "UML as an Ontology Modeling Language," *Proc. of the Workshop on Intelligent Information Integration, 16th Int. Joint Conference on AI (IJCAI-99)*, Stockholm, 1999
47. G. Guizzardi, H. Herre, G. Wagner: Towards ontological foundations for conceptual models. *ODBASE 2002*, LNCS vol. 2519, pp. 1100–1117
48. V Kabilan, P Johannesson, Semantic Representation of Contract Knowledge using Multi-Tier Ontology , to be published in the proceedings of Semantic Web and Databases workshop, VLDB 2003

WMS 2003 PC Co-chairs' Message

The idea of a Workshop on Metadata for Security (WMS) first came out at a EU project meeting held in Ispra about digital identities, where Robert Meersman was invited as a guest speaker. During that meeting, some researchers in the field of security and identity management exchanged ideas with experts in knowledge management and representation. Being part of both communities, I realized the high potential impact of this interdisciplinary dialogue and felt encouraged to accept Robert's suggestion of organizing a workshop on metadata security, where the same kind of fruitful discussion could be fostered by presentations of high quality papers.

Of course, there is nothing new in the increasing importance of knowledge-based metadata in the design and implementation of complex systems. Metadata have become since long a cornerstone of intelligent Web-based environments, including knowledge management, e-learning and content management systems. Web-based applications increasingly rely on metadata expressing properties like structure, nature and purpose of network resources, as well as of the users requesting them, in terms of domain ontologies. Also, novel metadata formats are playing a particularly important role in semantics-aware multimedia processing. Having said this, I would like to add that I consider security and identity management among the most interesting challenges for advanced metadata research. A new generation of security and privacy models and languages is needed, capable to fully exploit semantics representations. Also, innovative techniques are being developed aimed at protecting metadata from being disclosed or tampered with, especially in mobile computing environments, while preserving efficiency.

The aim of the Workshop on Metadata Security is offering to the international research community working on these and related issues a forum where research results can be presented, fostering multidisciplinary discussion on the role of metadata in security research. Looking at the papers collected in the WMS Proceedings as the result of a rigorous refereeing process, I am confident that this goal has been fully achieved while preserving the spirit of the informal, interdisciplinary exchange of ideas we had in Ispra. I would therefore like to thank all the members of the Program Committee, namely Elizabeth Chang, Sabrina De Capitani di Vimercati, Tharam Dillon, Eduardo Fernandez-Medina, Csilla Farkas, Christian Geuer-Pollmann, Phillip Hallam-Baker, Willem Jonker, Michiharu Kudo, Pierangela Samarati, Robert Tolksdorf, Xin Wang and Duminda Wijesekera for their great work in selecting the papers, and all the papers' authors for their high quality work.

August 2003

Ernesto Damiani, Università di Milano, Italy

(Workshop on Java Metadata for Security 2003 Program Committee Chair)

Efficient Processing of Secured XML Metadata

Ling Feng¹ and Willem Jonker^{1,2}

¹ Dept. of Computer Science, University of Twente, The Netherlands

`{ling,jonker}@cs.utwente.nl`

² Philips Research, The Netherlands

`willem.jonker@philips.com`

Abstract. Metadata management is a key issue in intelligent Web-based environments. It plays an important role in a wide spectrum of areas, ranging from semantic explication, information handling, knowledge management, multimedia processing to personalized service delivery. As a result, security issues around metadata management needs to be addressed in order to build trust and confidence to ambient environments. The aim of this paper is to bring together the worlds of security and XML-formatted metadata management in such a way that, on the one hand the requirement on secure metadata management is satisfied, while on the other other hand the efficiency on metadata processing can still be guaranteed. To this end, we develop an effective approach to enable efficient search on encrypted XML metadata. The basic idea is to augment encrypted XML metadata with encodings which characterize the topology and content of every tree-structured XML metadata, and then filter out candidate data for decryption and query execution by examining query conditions against these encodings. We describe a generic framework consisting of three phases, namely, *query preparation*, *query pre-processing* and *query execution*, to implement the proposed search strategy.

Keywords: Security, metadata, XML, encryption, search

1 Introduction

Ambient intelligence is an important theme in today's industrial and public research [3,9]. A key issue towards ambient intelligence is metadata management. From semantic explication, information handling, knowledge management, multimedia processing to personalized service delivery, metadata plays an important role. With XML [4] becoming the dominant standard for describing and interchanging data between various systems and databases on the Internet, Web-based applications nowadays increasingly rely on XML metadata to convey machine-understandable information and provide interoperability across the Web.

With this sheer volume of metadata flowing throughout ambient environments, the need to protect XML metadata content from being disclosed or tampered with is growing. One prototypical technique for building security and trust

is to distribute and store these metadata in encrypted form [12]. W3C recommends an “XML Encryption Syntax” to allow the encryption of XML data using a combination of symmetric and public keys, where element content is encrypted by means of a symmetric key that in turn is encrypted by means of the public key of the recipient [6,5]. Nevertheless, securing metadata in ciphertext should not hinder its processing for various applications. The deployed metadata security techniques should on the one hand satisfy the security requirements on XML metadata, while at the same time allow efficient manipulation of metadata without loss of confidentiality.

Since search is one of the basic operations that are carried out on metadata, a first step to proceed is to address the issue around effective and efficient searching for information in encrypted XML data. A straightforward approach to search on encrypted XML data is to decrypt the ciphertext first, and then do the search on the clear decrypted XML data. However, this inevitably incurs a lot of unnecessary decryption efforts, leading to a very poor query performance, especially when the searched data is huge, while the search target comes only from a small portion of it. To solve this problem, we ask two questions here: (1) “*Can we discard some non-candidate XML data and decrypt the remaining ones instead of the whole data set?*” (2) “*If so, how can we effectively and efficiently distinguish candidate XML data from non-candidate ones?*”

There is some previous work in different research areas that are related to our work. [13] presents techniques to support *keyword*-based search on encrypted textual string. Recently, [10,11] explore techniques to execute *SQL*-based queries over encrypted relational tables in a database-service provider model, where an algebraic framework is described for query rewriting over encrypted attributed representation. However, compared to the problem to be addressed in this study, the functionalities provided by the above work are still very limited and insufficient in performing complex *XPath*-based XML queries over encrypted semistructured XML data.

In this paper, we propose to augment encrypted XML metadata with *encodings* which characterize the topology and content of every encrypted tree-structured XML data. Here, a hash-based approach is employed to compute hashed XML path information into encoding schemas. The filtering of non-candidate data set can then be performed by examining query conditions, expressed in terms of *XPath* expressions [7], against the encodings. We outline a generic framework for conducting efficient queries on encrypted XML data, which is comprised of three phases, namely, *query preparation*, *query pre-processing*, and *query execution*. The *query preparation* phase aims to prepare for efficient query answering by encoding XML DTDs and XML documents before they are encrypted and stored in the database. This phase runs off-line. When a query is issued at runtime, the *query pre-processing* phase filters out impossible query candidates, so that the decryption and query execution that the *query execution* phase undertakes can be more focused on potentially target XML data.

The remainder of the paper is organized as follows. Section 2 outlines the framework for efficient querying over encrypted XML metadata. Section 3 and

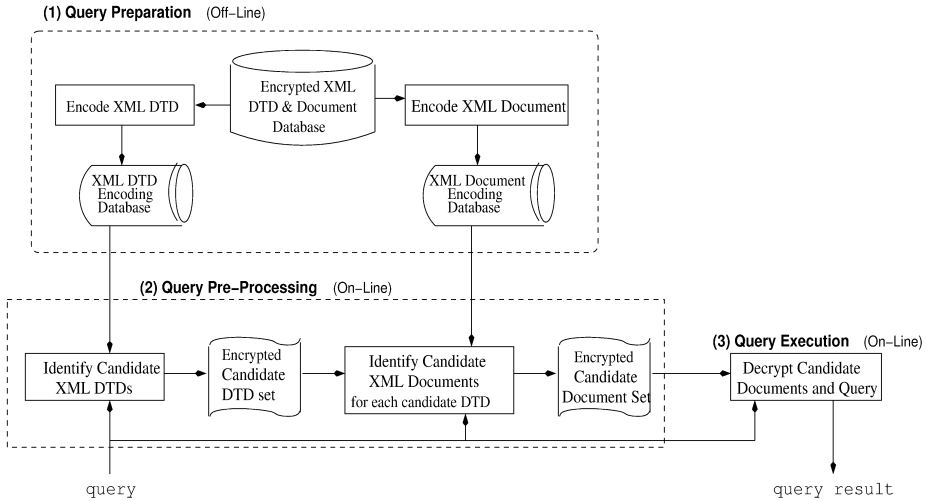


Fig. 1. A framework for querying over encrypted XML data

Section 4 describe in detail the query preparation phase and query pre-processing phase, respectively. Section 5 concludes the paper.

2 A Framework for Searching Encrypted XML Metadata

We assume the availability of DTD for each XML document in this study. For security reason, all XML DTDs and XML documents are encrypted and stored in a database. Figure 1 shows a generic framework for conducting efficient queries on encrypted XML data. It is comprised of three phases: *query preparation*, *query pre-processing* and *query execution*.

– Phase-1 (query preparation)

The aim of this phase is to prepare for efficient querying over encrypted XML data by encoding each XML DTD and associated documents before they are encrypted and stored in the database. Such an encoding is carried out in two steps, “*Encode XML DTD*” and “*Encode XML Document*”. The coding results of XML DTDs and documents are stored in two separate databases, called “*XML DTD Encoding Database*” and “*XML Document Encoding Database*”. In response to a query at run-time, these encodings can be used to pre-select **potential target documents** without the need to decrypt the whole document set in the database. We call these potential target documents **candidate documents** in the paper. Detailed encoding schemas and their computation will be described in Section 3.

– Phase-2 (query pre-processing)

It is obvious that decrypting all encrypted XML documents to answer a query inevitably incurs an excessive overhead, especially when the target

data constitutes a small portion of the database. In order to make encrypted XML query processing truly practical and computationally tractable, and meanwhile preserve security, for each query, we incorporate a pre-processing stage, whose aim is to filter out impossible candidates so that decryption and query execution can be more focused on potentially target documents. Two steps are conducted in this phase. First, a set of candidate XML DTDs are identified through the step “*Identify Candidate XML DTDs*”, which examines query conditions, expressed in terms of XPath expressions [7,8], against DTD encodings in the “*DTD Encoding Database*”. Then, corresponding to each selected candidate DTD, the “*Identify Candidate XML Documents*” step further filters out candidate documents based on documents’ encodings in the “*Document Encoding Database*”. The candidate DTD set and document set returned are subsets of the original encrypted DTD set and document set, respectively. Detailed descriptions of these two steps will be given in Section 4.

– **Phase-3 (query execution)**

The identified candidate DTDs and documents, returned from Phase-2, are decrypted into clear DTDs and documents, on which the query can be executed. Here, conventional XML query engines can be employed in this phase. As querying over non-encrypted XML data has been widely investigated in the literature, we focus our study on Phase-1 and Phase-2 in this study.

3 Query Preparation Phase

In this section, we propose a hash-based strategy to encode encrypted XML data for the query preparation phase. Based on the encodings obtained, the query pre-processing phase, which will be addressed in Section 4, can then effectively filter out query candidates, i.e., potential targets, from among a large set of blind documents in the database. Due to different characteristics and functions of XML DTDs and documents, we encode XML DTDs and XML documents separately using different encoding schemas.

In the following, we first describe the computation method for encoding XML DTDs, followed by the method for encoding XML documents. For ease of explanation, a running example shown in Fig. 2 is used throughout the discussion. A graphical representation of the DOM tree structure of the example DTD, DTD_1 , and the example document is outlined in Fig. 2(c).

3.1 Encoding XML DTDs

An XML DTD defines the legal building blocks of its conforming XML documents, like what elements, attributes, etc. are permitted in the documents [4]. These components construct a hierarchical tree structure that underlies the contents of the documents, with each path of the tree addressing a certain part of an document.

```

<!DOCTYPE payInfo [
  <!ELEMENT payInfo (creditCard?, amount+)>
  <!ELEMENT creditCard (number, name, address)>
  <!ATTLIST creditCard limit CDATA #IMPLIED>
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT amount (#PCDATA)>
]

```

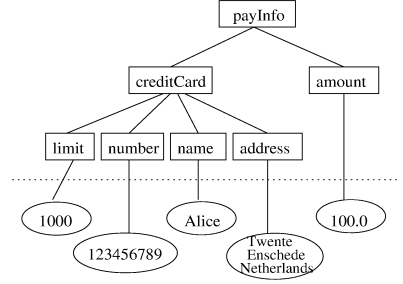
(a) An XML DTD exmple – DTD1

```

<payInfo>
  <creditCard limit=1000>
    <number> 123456789 </number>
    <name> Alice </name>
    <address> Twente 7500 AE, Netherlands </address>
  </creditCard>
  <amount> 100.0 </amount>
</payInfo>

```

(b) An XML document exmple that conforms to DTD1



(c) A graphical representation of the DOM tree structure of DTD1 with the example document

Fig. 2. A running example of an XML document with its DTD

As the query pre-processing phase works on the basis of XPath expressions embedded in a query like XQuery, to prepare for efficient candidate selection, we thus take the strategy to encode each XML DTD in the unit of *path*. The notions of *path* and *path length* are defined as follows.

Definition 1. A *path* p is a sequence of nodes $n_1, n_2, \dots, n_k$, denoted as $p = (n_1/n_2/\dots/n_k)$, where for any two consecutive nodes, n_i and n_{i+1} ($1 \leq i \leq k-1$, $k \geq 1$), there exists an edge between them.

The *length* of path p , denoted as $|p|$, is the total number of edges in the path. That is, $|p| = (n_1/n_2/\dots/n_k) = k-1$. $\square$

Table 1 lists all the paths, which are of various lengths, extracted from the example DTD DTD_1 in Fig. 2. Here, the content nodes under the dotted line are exempt from consideration, since they do not appear in the DTD.

In essence, we use the technique of hashing on each path of an XML DTD to compose DTD encodings. Paths of different lengths will be hashed into different hash tables named $DTDHashTable_0, DTDHashTable_1, DTDHashTable_2, \dots, DTDHashTable_{max\_pathLen}$, respectively. All paths of length l (where $1 \leq l \leq max\_pathLen$), no matter which DTD it comes from, will share one single hash table $DTDHashTable_l$, with each bucket indicating a set of DTDs, whose paths have been hashed into the bucket. Suppose we have a path p extracted from DTD_1 , the hash function $HashFunc(p)$ computes its hash value, i.e., bucket address in the hash table $DTDHashTable_{|p|}$. (Detailed computation of hash values will be given shortly.) We mark the corresponding bucket entry with an indicator of DTD_1 , signifying the DTD where p locates.

To filter out non-candidate DTDs for a query, we compute the hash values for all XPath expressions in the query using the same hash function, and then check the corresponding buckets in the DTD hash tables to obtain a subset of DTDs that

Table 1. Paths extracted from the example XML DTD

Path length	Path
2	$p_1 = (\text{payInfo}/\text{creditCard}/\text{limit})$
	$p_2 = (\text{payInfo}/\text{creditCard}/\text{number})$
	$p_3 = (\text{payInfo}/\text{creditCard}/\text{name})$
	$p_4 = (\text{payInfo}/\text{creditCard}/\text{address})$
1	$p_5 = (\text{payInfo}/\text{creditCard})$
	$p_6 = (\text{payInfo}/\text{amount})$
	$p_7 = (\text{creditCard}/\text{limit})$
	$p_8 = (\text{creditCard}/\text{number})$
	$p_9 = (\text{creditCard}/\text{name})$
	$p_{10} = (\text{creditCard}/\text{address})$
0	$p_{11} = (\text{payInfo})$
	$p_{12} = (\text{creditCard})$
	$p_{13} = (\text{amount})$
	$p_{14} = (\text{limit})$
	$p_{15} = (\text{number})$
	$p_{16} = (\text{name})$
	$p_{17} = (\text{address})$

Algorithm 1 Hash function $HashFunc(p)$

Input: path $p = (n_1/n_2/\dots/n_k)$, a fixed size s for node names,
hash table size $SizeDTDHashTable_{|p|}$;

Output: hash value of p

- For each node n_i ($1 \leq i \leq k$), chop its name uniformly into an s -letter string
 $ChopName(n_i, s) = x_{n_{i,1}}x_{n_{i,2}}\dots x_{n_{i,s}}$,
where $x_{n_{i,1}}, x_{n_{i,2}}, \dots, x_{n_{i,s}}$ are letters in the name string of node n .
- For each s -letter node name $x_{n_{i,1}}x_{n_{i,2}}\dots x_{n_{i,s}}$, convert it into a decimal integer
 $Base26ValueOf(x_{n_{i,1}}x_{n_{i,2}}\dots x_{n_{i,s}}) =$
 $offset(x_{n_{i,1}}) * 26^{s-1} + offset(x_{n_{i,2}}) * 26^{s-2} + \dots + offset(x_{n_{i,s}}) * 26^0 = V_{n_i}$,
where $offset(x_{n_{i,j}})$ ($1 \leq j \leq s$) returns the position of letter $x_{n_{i,j}}$ among 26 letters.
- Compute hash value of $p = (n_1/n_2/\dots/n_k)$
 $HashFunc(n_1/n_2/\dots/n_k) =$
 $(V_{n_1} * 10^{k-1} + V_{n_2} * 10^{k-2} + \dots + V_{n_k} * 10^0) \bmod SizeDTDHashTable_{|p|}$.

possibly contain the requested paths. These DTDs are candidate DTDs to be considered for the query.

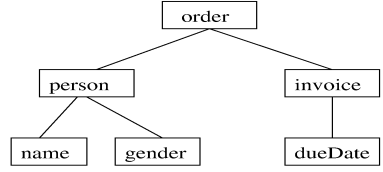
Algorithm 3.1 elaborates the procedures in computing the hash value for path $p = (n_1/n_2/\dots/n_k)$. It proceeds in the following three steps.

First, node names in path p which could be of different lengths are uniformly chopped into the same size s , given by users as an input parameter,

```

<!DOCTYPE payInfo [
  <!ELEMENT order (person, invoice)>
  <!ELEMENT person (name, gender)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT gender (#PCDATA)>
  <!ELEMENT invoice (dueDate)>
  <!ELEMENT dueDate (#PCDATA)>
]>

```



(a) Another DTD example – DTD2

(b) A tree structure of DTD2

Fig. 3. Another DTD example with its DOM tree-structure

through the function *ChopName* (Algorithm 3.1, line 1). For example, let $s=4$, $\text{ChopName}(\text{"creditCard"}, 4) = \text{"cred"}$, $\text{ChopName}(\text{"payInfo"}, 4) = \text{"payI"}$, $\text{ChopName}(\text{"name"}, 4) = \text{"name"}$.

Second, the chopped node name strings which are of fixed size after Step 1 are further converted into decimal integers via function *Base26ValueOf* (Algorithm 3.1, line 2). Example 1 explicates how it works when the size of node name string is set to 4.

Example 1. When we let a 4-letter node name $x_1x_2x_3x_4$, which are case insensitive, represent a base-26 integer, we let the letter “a” represent the digit-value 0, the letter “b” represent the digit-value 1, the letter “c” represent the digit-value 2, the letter “d” represent the digit-value 3, and so on, up until the letter “z”, which represents the digit-value 25. Given a letter, function *offset* returns such a digit-value. The 4-letter node name $x_1x_2x_3x_4$ can thus be converted into a decimal integer using the formula: $\text{Base26ValueOf}(x_1x_2x_3x_4) = \text{offset}(x_1) * 26^3 + \text{offset}(x_2) * 26^2 + \text{offset}(x_3) * 26^1 + \text{offset}(x_4) * 26^0$.

Assume that $x_1x_2x_3x_4 = \text{"name"}$, since the digit-values of “n”, “a”, “m” and “e” are $\text{offset}(\text{"n"}) = 13$, $\text{offset}(\text{"a"}) = 0$, $\text{offset}(\text{"m"}) = 12$, and $\text{offset}(\text{"e"}) = 4$ respectively, we have $\text{Base26ValueOf}(\text{"name"}) = 13 * 26^3 + 0 * 26^2 + 12 * 26^1 + 4 * 26^0 = 13 * 17576 + 0 + 312 + 4 = 228802$.

In a similar way, we have $\text{Base26ValueOf}(\text{"cred"}) = 2 * 26^3 + 17 * 26^2 + 4 * 26^1 + 3 * 26^0 = 2 * 17576 + 17 * 676 + 104 + 3 = 35152 + 11492 + 104 + 3 = 46751$. $\square$

A general calculation of *Base26ValueOf* is: $\text{Base26ValueOf}(x_1x_2 \dots x_s) = \text{offset}(x_1) * 26^{s-1} + \text{offset}(x_2) * 26^{s-2} + \dots + \text{offset}(x_s) * 26^0$.

Finally, hash function *HashFunc* derives the hash value of $p = (n_1/n_2/\dots/n_k)$ based on the value V_{n_i} returning from function *Base26ValueOf* on each node n_i (Algorithm 3.1, line 3). $\text{HashFunc}(n_1/n_2/\dots/n_k) = (V_{n_1} * 10^{k-1} + V_{n_2} * 10^{k-2} + \dots + V_{n_k} * 10^0) \bmod \text{SizeDTDHashTable}_{|p|}$

In order to provide a more complete overview on the hash-based encoding method, we introduce another DTD example DTD_2 as shown in Fig. 3. Using the same hash function, Fig. 4 illustrates the hash results for all the paths from DTD_1 and DTD_2 .

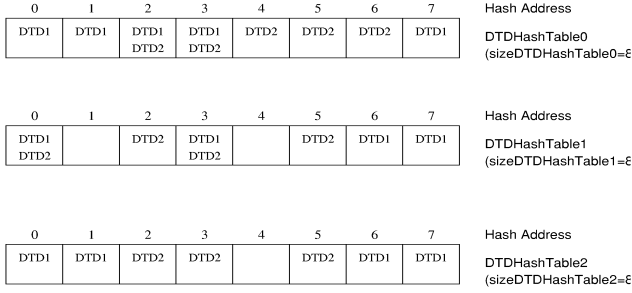


Fig. 4. Encodings of the example DTD_1 and DTD_2 ($DTDHashTable_0$, $DTDHashTable_1$ and $DTDHashTable_2$)

Table 2. Pairs of element/attribute with content/value in the example XML document, together with their hash and mapped values

(Element/Attribute c_{name} , Content/Value c_{val})	$HashFunc(c_{name})$	$MapFunc(c_{val})$
$c_1 = (\text{limit}, 1000)$	0	1
$c_2 = (\text{number}, 123456789)$	1	10
$c_3 = (\text{name}, \text{"Alice"})$	0	0
$c_4 = (\text{address}, \text{"Twente, Enschede, Netherlands"})$	2	25
$c_5 = (\text{amount}, 100.0)$	1	7

3.2 Encoding XML Documents

XML documents that conform to one XML DTD possess a similar structure, but with possibly different element contents and/or attribute values to distinguish different documents. For instance, one conforming document of the example DTD shown in Fig. 2 has a *limit* attribute of value 1000, represented as *limit*=1000 for simplicity. Its elements *number*, *name*, *address* and *amount* have contents 123456789, “Alice”, “Twente, Enschede, Netherlands” and 100.0, respectively.

After encoding XML DTDs, i.e., all possible paths with each containing a sequence of nodes corresponding to elements or attributes, the second task of the query preparation phase is to encode their conforming documents, i.e., all pairs of element and element content (*element*, *element content*), attribute and attribute value (*attribute*, *attribute value*). Due to the different nature of contents, encoding documents is conducted in a different way from encoding DTDs, with the result stored in the “*Document Encoding Database*”. In the following, we describe the method of encoding a pair, $c = (c_{name}, c_{val})$ (where c_{name} denotes the element/attribute, and c_{val} denotes the corresponding element content/attribute value), into a hash table named *DOCHashTable*.

We adopt the *separate chaining* strategy to resolve hashing collision for *DOCHashTable*. That is, we place all pairs that collide at a single hash address on a linked list starting at that address. The hash address of each pair is calculated via function $HashFunc(p)$ (Algorithm 3.1), using a different hash

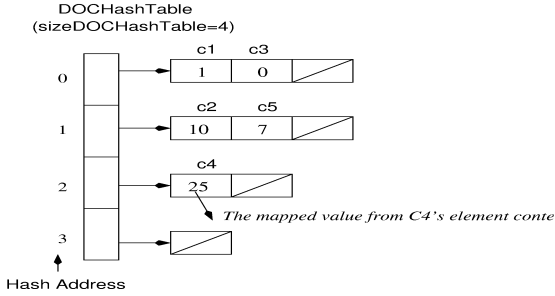


Fig. 5. Encodings of the example document (*DOCHashTable*)

table size, which is *SizeDOCHashTable* rather than *SizeDTdHashTable*_{|p|}. In this case, path *p* always contains only one node, which is $p = (c_{name})$ and $|p| = 0$. For example, let $s=4$, and the size of hash table *DOCHashTable* equal to 4 (i.e., $SizeDOCHashTable = 4$). We have $ChopName("limit") = "limi"$. $Base26ValueOf("limi") = 11 * 26^3 + 8 * 26^2 + 12 * 26 + 8 = 199064$, $HashFunc(limit) = 199064 * 10^0 \bmod 4 = 0$.

After the derivation of bucket address in the hash table *DOCHashTable* from c_{name} , the entry to be put into the corresponding bucket is computed based on c_{val} , using the technique developed in [10]. The basic idea is to first divide the domain of node c_{name} into a set of *complete* and *disjoint* partitions. That is, these partitions taken together cover the whole domain; and any two partitions do not overlap. Each partition is assigned a *unique* integer identifier. The value c_{val} of element/attribute node c_{name} is then mapped to an integer, corresponding to the partition where it falls [10]. For example, we can partition the domain of attribute *limit* into $[0, 500]$, $(500, 1000]$, $(1000, \infty]$ of identifier 0, 1, 2, respectively. The *limit* value 1000 is thus mapped to integer 1, and stored in the first bucket of *DOCHashTable*, since $HashFunc(limit) = 0$. The hash values for other pairs in the example document are calculated in the same way, which are shown in Table 2.

Note that the partition of a domain can be done based on the *semantics* of data and relevant applications. For instance, we can categorize the domain of element *name* according to the alphabetical order. The domain of element *address* can be partitioned according to *province* or *country* where located. For simplicity, in the current study, we enforce *order preserving* constraint on such a mapping " $MapFunc : domain(c_{name}) \rightarrow Integer$ ", which means that for any two values c_{val_1} and c_{val_2} in the domain of c_{name} , if $(c_{val_1} \leq c_{val_2})$, then $MapFunc(c_{val_1}) \leq MapFunc(c_{val_2})$.

Assume the mapping functions for *number*, *name*, *address* and *amount* return identifiers, as indicated in Table 2. Figure 5 plots the resulting encoding, i.e., *DOCHashTable*, for the example XML document given in Fig. 2.

4 Query Pre-processing Phase

The aim of the query pre-processing phase is to identify candidate DTDs and documents by checking the query against the encodings of DTDs and documents, obtained after the query preparation phase. In this section, we first provide a brief description of XPath expressions used in query representation. We then discuss a method to match such XPath expressions to paths as described in Section 3 in order to facilitate candidate DTD and document selection. A two-step procedure is finally illustrated to identify candidate DTDs, followed by candidate documents for each selected candidate DTD.

4.1 XPath Expressions

The XPath language is a W3C proposed standard for addressing parts of an XML document [7]. It treats XML documents as a tree of nodes corresponding to elements/attributes, and offers an expressive way to specify and locate nodes within this tree.

XPath expressions state structural patterns that can be matched to paths, consisting of a sequence of nodes in the XML data tree [2,1]. Such paths can be either absolute paths from the root of the data tree, or relative one starting with some known context nodes. The hierarchical relationships between the nodes are specified in XPath expressions using parent-child operator (“/”) and ancestor-descendant operator (“//”). For example, the XPath expression “/payInfo/creditCard/@limit” addresses *limit* attribute of *creditCard* which is a child element of the *payInfo* root element in the document. The *name* element in the relative path expression “//creditCard/name” is a child relative to its parent *creditCard* element. The expression “/payInfo//name” addresses *name* descendant element of the *payInfo* root element.

XPath also allows the use of a wildcard operator (“*” or “@*”), which can match any element or attribute node with respect to the context node in the document data tree. In addition, predicates, enclosed in square brackets (“[]”), can also be applied to further refine the selected set of nodes in XPath expressions. For example, “/payInfo/creditCard[@limit < 1000]/name” selects the *name* elements of the XML document if the attribute *limit* of *creditCard* has a value less than 1000.

Operators like (“|”) and (“and”) can also be applied to select constituent nodes of paths [7]. For instance, “/payInfo/(creditCard|cash)/name” expression selects every *name* element that has a parent that is either a *creditCard* or a *cash* element, that in turn is a child of a root element *payInfo*. On the contrary, “/payInfo/creditCard[@limit and @dueDate]” indicates all the *creditCard* children of the root element *payInfo* that must have both a *limit* attribute and a *dueDate* attribute.

4.2 Mapping XPath Expressions to Paths

Considering that DTD encodings are computed against *paths* as defined in Definition 1 in Section 3, for efficient encoding-based query candidate pre-selection,

we first need to match an *XPath expression* e , which is used to locate parts of a data tree, to a set of *paths* through the following three steps.

Step 1. Decompose XPath expression e into several ones at the point of “//” operator.

Since paths to be encoded by the offline query preparation phase have only parent-child relationships (“/”) between two consecutive nodes (as shown in Table 1), we break an XPath expression from the points where the “//” operator locates, into several ones where each node, except for the first one, is prefixed only by “/”. The resulting XPath expressions thus contain no ancestor-descendant relationships (“//”) between every two consecutive nodes.

For ease of explanation, we signify the XPath expressions derived after Step 1 using a prime symbol like e' . They form the input of Step 2.

Step 2. Simplify predicate constraints in each XPath expression e' to only hierarchical relationships.

As DTD encoding relieves value constraints on path nodes, and focuses only on their hierarchical relationships, to facilitate candidate DTD filtering based on path encodings, we relax value constraints on nodes like “[*amount* > 100]” and “[*@limit* = 1000]”, specified in XPath predicate expressions, and keep only their inherent parent-child or element-attribute relationships.

Let e'' denote an XPath expression returned after Step 2.

Step 3. Eliminate logical “|” and “and” operators in each XPath expression e'' by rewriting the expression into several ones logically connected with “^” or “v”.

To match the notion of path in Definition 1, every XPath expression after Step 2 which contains the logical operators “|” and “and” is substituted by a set of shorter XPath expressions, which are logically connected via “^” or “v”.

After undergoing the above three steps, an original XPath expression is transformed into a set of **simple XPath expressions**, which contain no ancestor-descendant relationships between two consecutive nodes, no value constraints on nodes, and no logical operators (“|”) and (“and”). Each such kind of simple XPath expressions corresponds to a path defined in Definition 1.

Example 2. From an original XPath expression “/payInfo[*amount* > 100]//name”, we can derive two simple XPath expressions “/payInfo/*amount*” ^ “/name”.

An XPath expression with a predicate constraint and operator (“|”) like “/payInfo[*amount* > 100]/(creditCard|cash)/name” leads to three simple XPath expressions which are: “/payInfo/*amount*” ^ (“/payInfo/creditCard/name” v “/payInfo/cash/name”). □

4.3 Identification of Candidate DTDs and Documents

On the basis of simple XPath expressions generated from XPath expressions embedded in a query, we can now define the concepts of candidate DTDs and documents for the given query.

An XML DTD is called a **candidate DTD** for a query, if for every simple XPath expression derived from the query, there *possibly* exists a path p in the DTD, that matches this simple XPath expression.

In a similar fashion, we define that an XML document is a **candidate document** for a query, if and only if: 1) its DTD is a candidate DTD¹; and 2) it *possibly* satisfies all predicate constraints on the nodes inside all the XPath expressions embedded in the query.

The pre-selection of potential query targets starts with the identification of candidate DTDs, followed by the identification of candidate documents under each candidate DTD that has been identified.

I. Identify Candidate DTDs by Hashing Paths

Given a query, to check out which encrypted DTDs are candidate DTDs for each simple XPath expression generated from the query, we match it to a path p , and compute the hash value for p using the same hash function $HashFunc(p)$ (Algorithm 3.1) while encoding the DTDs. According to the hash value (i.e., bucket address) returned, we consult with the corresponding bucket in the hash table $DTDHashTable_{|p|}$, which gives all the DTDs that may possibly contain path p . The rationale for this is straightforward: *if path p is present in the DTD, it will be hashed to the bucket in $DTDHashTable_{|p|}$, leaving a mark for this DTD in the bucket entry.*

Example 3. Suppose a query consists of only one simple XPath expression, corresponding to the path $p = (payInfo/creditCard/dueDate)$. Referring to the DTD encoding schema illustrated in Fig. 4, where $s = 4$ and

$SizeDTDHashTable_2 = 8$, its hash value is computed as follows:

Step 1: $ChopName("payInfo", 4) = "payI"$, $ChopName("creditCard", 4) = "cred"$, $ChopName("dueDate", 4) = "dueD"$.

Step 2: $Base26ValueOf("payI") = 264272$, $Base26ValueOf("cred") = 46751$, $Base26ValueOf("dueD") = 66355$.

Step 3: $HashFunc(payInfo/creditCard/dueDate) = (Base26ValueOf("payI") * 10^2 + (Base26ValueOf("cred") * 10^1 + Base26ValueOf("dueD") * 10^0) \bmod SizeDTDHashTable_2 = (264272 * 100 + 46751 * 10 + 66355) \bmod 8 = 1$

Due to its hash value 1, we can be sure that the example DTD_2 does not contain that path, since the entry at address 1 in $DTDHashTable_2$ only signifies DTD_1 . As a result, only DTD_1 will be returned as the candidate DTD, DTD_2 and its associated documents can thus be discarded from the further search. $\square$

II. Identify Candidate Documents by Hashing Element/Attribute and Content/Value Pairs

After pre-selecting the candidate DTD set for the given query, we are now in the position to filter out candidate documents underneath each candidate DTD. At this stage, various value constraints in the form of $[c_{name} \theta c_{val}]$ (where c_{name} denotes the name of an element/attribute node, θ is one of the operators in $\{=, \neq, <, \leq, >, \geq\}$, and c_{val} denotes the element content/attribute value) on

¹ Recall that we assume the availability of DTD for each document in this study.

path nodes are taken into consideration. Clearly, a candidate document must not violate any of the value constraints specified within the XPath expressions in the query. We perform such kind of examination based on the document encodings (i.e., *DOCHashTable*).

Taking the constraint $[c_{name} \theta c_{val}]$ for example, we first hash the node name c_{name} (i.e., a path containing only one node) into *DOCHashTable* via hash function $HashFunc(c_{name})$. Meanwhile, we also calculate the range identifier of c_{val} using the order preserving function $MapFunc(c_{val})$. Finally, we compare each entry value v linked to the bucket address $HashFunc(c_{name})$ in *DOCHashTable*: if $\exists v (v \theta MapFunc(c_{val}))$, then the constraint $[c_{name} \theta c_{val}]$ possibly holds.

Example 4. Assume a query embeds an XPath expression “/payInfo/creditCard [@limit > 2000]/name”, which enforces a constraint $[@limit > 2000]$ on *creditCard* element. Referring to the document encoding schema in Fig. 5, where $s = 4$ and $SizeDOCHashTable = 4$. We have $HashFunc(limit) = 0$ and $MapFunc(2000) = 2$. Since all the entries at address 0 in *DOCHashTable* are either 1 or 0, which is not greater than 2 ($= MapFunc(2000)$), therefore, the example document is not a candidate document for this query, and can thus be discarded. $\square$

5 Conclusion

In this paper, we employ the hash technique to compute encodings associated with each encrypted XML metadata to allow effective pre-filtering of non-candidate data for a given query, expressed in terms of XPath expressions. We are currently conducting experiments to investigate the performance of the proposed strategy.

References

1. M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. the 26th Intl. Conf. on Very Large Data Bases*, pages 53–64, Cairo, Egypt, September 2000.
2. C. Chan, P. Felber, M. Carofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proc. the Intl. Conf. on Data Engineering*, California, USA, February 2002.
3. European Commission. Scenarios for ambient intelligence in 2010. <http://www.cordis.lu/ist/istag.htm>, 2001.
4. World Wide Web Consortium. Extensible markup language (XML) 1.0. <http://www.w3.org/TR/REC-xml>, October 2000.
5. World Wide Web Consortium. XML encryption requirements. <http://www.w3.org/TR/xml-encryption-req>, March 2002.
6. World Wide Web Consortium. XML encryption syntax and processing. <http://www.w3.org/TR/xmlenc-core/>, August 2002.

7. World Wide Web Consortium. XML path language (XPath) 2.0.
<http://www.w3.org/TR/xpath20/>, november 2002.
8. World Wide Web Consortium. XQuery 1.0: an XML query language.
<http://www.w3.org/TR/xquery/>, november 2002.
9. E. Dijkstra, W. Jonker, and H. van Gageldonk. Data and content management (chapter). In *The New Everyday – Views on Ambient Intelligence*, E. Aarts and S. Marzano (eds.), Koninklijke Philips Electronics N.V., ISBN 90-6450-502-0, 2003.
10. H. Hacigümüş, B. Lyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. the ACM SIGMOD Intl. Conf. on Management of Data*, pages 216–227, Wisconsin, USA, June 2002.
11. H. Hacigümüş, B. Lyer, and S. Mehrotra. Providing database as a service. In *Proc. Intl. Conf. on Data Engineering*, 2002.
12. W. Jonker. XML and secure data managment in an ambient world. *Computer Systems Science & Engineering (to appear)*, 2003.
13. D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. the IEEE Symposium on Security and Privacy*, 2000.

Multimedia Access Control Using RDF Metadata^{*}

Naren Kodali¹, Csilla Farkas^{3,4}, and Duminda Wijesekera^{1,2}

¹ Dept of Info. and Software Eng.,

² Center for Secure Information Systems,
George Mason University, Fairfax, VA 22030-4444,
{nkodali,dwijesek}@gmu.edu

³ Information Security Laboratory,

⁴ Dept of Computer Science and Eng.,
University of South Carolina, Columbia, SC-29208,
farkas@cse.sc.edu

Abstract. The Synchronized Multimedia Integration Language (SMIL) [Aya01] is an W3C [W3C03] specification for authoring multimedia documents. Although SMIL has XML like syntactic constructs, unlike XML, SMIL compositions have an intended interpretation stemming from intuitive notions of playing out many media streams relative to each other. Thus, more than one SMIL syntactic expression can represent a multimedia composition with the same intended semantics. In this work we propose a normal form for SMIL objects that allows to specify security policies that are independent of representational syntax. We also show how to represent access control and QoS policies applicable to multimedia compositions by decorating SMIL compositions with RDF [KC03] statements. Our RDF statements are based on an RDF structure tailored to represent known security paradigms such as Discretionary, Mandatory, and Role-Based Access Control. Once the security paradigm is chosen and the SMIL document is decorated with security and QoS specifications, we show how to create secure views of the SMIL document. We call these views secure normal forms. Next, we show how a secure multimedia server can use these views to provide secure runtime environment.

1 Introduction

SMIL [Aya01] is an XML-like language for authoring multimedia documents. Unlike XML for textual documents, SMIL constructs have an *intended meaning* that must be enforced by application runtimes. Therefore, any security policy specification has to respect that semantics. This paper proposes a framework to do so for a chosen fragment of SMIL. This fragment consists of SMIL specifications constructed using sequential (`<seq>`) and parallel (`<par>`) composition operators.

^{*} This work was partially supported by the National Science Foundation under grants CCS-0113515 and 0112874.

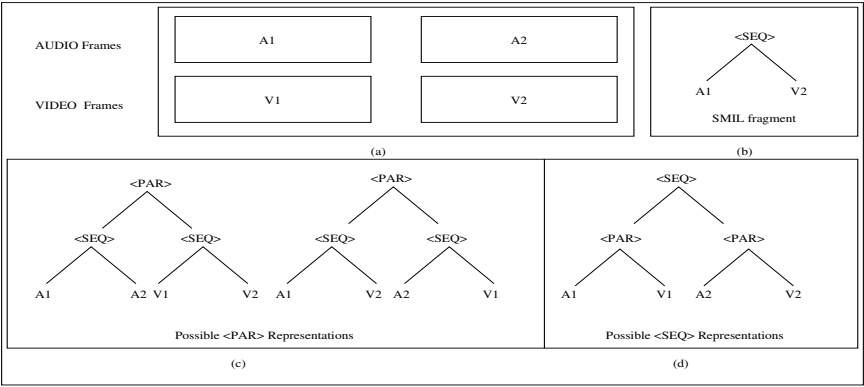


Fig. 1. Equivalence Class of the SMIL Constructs

Our framework uses two techniques. The first is to transform a SMIL document to a syntactic form that preserves the runtime semantics and shows the semantic hierarchy of any SMIL specification. We call this syntactic form the *SMIL normal form (smilNF)* of the document, and is structurally similar to the disjunctive normal form of a formula in propositional logic. We show that any arbitrary SMIL (syntax) tree does not accurately represent its complete semantic hierarchy as it exists today. We present a method to obtain the hierarchy from the normal form. It is our position that normal forms are necessary because security policies may depend on the object hierarchy and not necessarily on one of its syntactic representations.

We follow the specifications of the W3C in using the Resource Description Framework (RDF) [KC03,MM03] to define metadata for specifying security and QoS policies. In order to do so, we propose a preliminary form of an RDF structure to model security and QoS specifications for SMIL documents. Based on our structure, RDF decorations can be superimposed on SMIL documents in their normal form so that security and QoS specifications can be enforced by security and QoS aware runtimes. We now introduce our first issue by an example.

As described in more detail in Section 3, SMIL uses `<par>` and the `<seq>` to specify parallel and sequential playing of multimedia streams. In SMIL, basic objects are media intervals. A media interval begins at a specified time, plays out for a specified duration and consequently ends at a specified time. This constitutes a rudimentary semantics for media intervals such as (audio) A_1, A_2 and (video) V_1, V_2 in Fig. 1. In this semantics two streams are connected by a `<par>` if they begin and end playout at the same time. Two streams are connected by a `<seq>` if the second begins when the first ends.

Audio(A_1, A_2) and Video(V_1, V_2) frames as shown in part (a) of Fig. 1, can be represented in SMIL in atmost three different ways using the `<par>` and `<seq>` constructs as shown in Fig. 1 and explained below.

1. $\langle \text{par} \rangle \langle \text{seq} \rangle A_1, A_2 \langle / \text{seq} \rangle \langle \text{seq} \rangle V_1, V_2 \langle / \text{seq} \rangle \langle / \text{par} \rangle$
2. $\langle \text{par} \rangle \langle \text{seq} \rangle A_1, V_2 \langle / \text{seq} \rangle \langle \text{seq} \rangle A_2, V_1 \langle / \text{seq} \rangle \langle / \text{par} \rangle$
3. $\langle \text{seq} \rangle \langle \text{par} \rangle A_1, V_1 \langle / \text{par} \rangle \langle \text{par} \rangle A_2, V_2 \langle / \text{par} \rangle \langle / \text{seq} \rangle$
4. Because $\langle \text{par} \rangle$ is *commutative* $\langle \text{par} \rangle A_1, V_1 \langle / \text{par} \rangle$ is the same as $\langle \text{par} \rangle V_1, A_1 \langle / \text{par} \rangle$ and $\langle \text{par} \rangle A_2, V_2 \langle / \text{par} \rangle$ is the same as $\langle \text{par} \rangle V_2, A_2 \langle / \text{par} \rangle$.

Now consider the fragment $\langle \text{seq} \rangle A_1, V_2 \langle / \text{seq} \rangle$, as shown in part(b) is not a subtree of the given syntactic representations in part(d), but a sub-object of the SMIL tree. The identity of this *protection object* therefore is not a node in the XML tree, but an equivalence class, represented by the *normal form*.

Therefore, we propose that every SMIL specification is to be transformed to a sequence of parallel compositions that we call the *smil normal form (smilNF)* and show that all sub-objects of a SMIL object can be obtained as a subtree (created from) of this form. We also propose that security and QoS policies be specified on SMIL specifications in smilNF, and not on arbitrary syntax trees - because as shown, syntactic substructure does not coincide with semantic inheritance in SMIL.

Consequently, we present a nomenclature to specify security policies by appropriately decorating SMIL documents in smilNF. In order to do so, we have chosen the RDF [KC03,MM03] syntax. Because RDF syntax makes sense with respect to some RDF metadata, we propose a metastructure for specifying access control and QoS policies applicable to multimedia compositions. Here again, we have chosen to represent limited features of access control policies. We show how some rudimentary discretionary, mandatory (also called multilevel secure (MLS)) and role-based access control policies can be specified using our nomenclature.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes the SMIL syntax. Section 4 defines the object identity and the SMIL normal form. Section 5 describes secure normal forms and give two algorithms for conversion for the secure normal forms. Section 6 describes the proposed RDF metastructure. Section 7 describes how a secure run-time may communicate to obtain SMIL formatted data from a secure server. Section 8 concludes the paper.

2 Related Work

RDF is a W3C standard for representing metadata on the web. RDF provides syntax for representing entities, their properties and relationships. RDF Abstraction and Syntax [KC03], and RDF Primer [MM03] specify metainformation representation, and RDF Schema [BG03] is a general purpose schema language. Hayes et al. [Hay03] describes semantical aspects of RDF. We use the RDF vocabulary to specify our metastructure.

SMIL has a RDF based metainformation module [Mic01], but is insufficient to specify security policies. Independent of SMIL, Quality of Service (QoS) is an integral part of multimedia. Wijesekera et al. [WS96] proposed properties of quality metrics associated with continuous media and Gu et al. [GNY⁺01]

propose *HQML*, a language to negotiate some QoS parameters between clients and server.

We consider DAC, MLS and RBAC as security models governing the display and access to SMIL formatted multimedia. DAC(discretionary access control) is used to control access by restricting a subjects's access to an object. Sandhu et al [SS96], [SFK00] describe the principles and practices of RBAC systems. In RBAC the *role* that an user plays in the context of the application determines his access privileges. Multilevel security (MLS) systems provide controlled information flow based on the security classification of the protection objects (e.g., data items) and subjects of the MLS system (e.g., applications running in behalf of a user).

Damiani et al. [DdV03] discuss feature protection of XML format images where the primary focus is controlled dissemination of sensitive data within an image. They propose an access control model with complex filtering conditions. This model uses SVG to render the map of a physical facility. This model has limitations when compared to flexibility and adaptability to issues, such as temporal and operational semantics. Bertino et al. [BHAE02] propose a security framework to model access control in video databases. Their objects are sequences of frames or identifiable objects within a frame. Their actions are viewing and editing. However they do not explain how objects with controlled accesses are released so that they do not lose their runtime semantics.

The main difference between SMIL and other XML documents are the temporal synchrony and continuity of the latter. The process of retrieval without losing the sense of continuity and synchronization needs better techniques and algorithms which all of the above models do not completely address. Kodali et al. [KW02,KWJ03,KFW03] propose three different models for enforcing different security paradigms. A release control for SMIL formatted multimedia objects for pay-per-view movies on the Internet that enforces DAC is described in [KW02]. A secure and progressively updatable SMIL document [KWJ03] is used to enforce RBAC and respond to traffic emergencies. In an emergency response situation, different recipients of the live feeds have to be discriminated to people playing different roles. [KFW03] describes an MLS application for secure surveillance of physical facilities where guards with different security classification in charge of the physical security of the building are provided live feeds matching their level in the MLS subject hierarchy.

3 SMIL: Synchronized Multimedia Integration Language

SMIL [Aya01] is an extension to XML developed by W3C to author multimedia presentations with audio, video, text and images to be integrated and synchronized. The distinguishing features of SMIL over XML are the syntactic constructs for timing and synchronizing live and stored media streams with qualitative requirements. In addition, SMIL provides a syntax for spatial layout including non-textual and non-image media and hyperlinks. We do not address the later aspects of SMIL in this paper. Consequently we explain those SMIL constructs that are relevant for our application.

SMIL constructs for synchronizing media are $\langle \text{seq} \rangle$, $\langle \text{excl} \rangle$ and $\langle \text{par} \rangle$. They are used to hierarchically specify synchronized multimedia compositions. The $\langle \text{seq} \rangle$ element plays its children one after another in sequence. $\langle \text{excl} \rangle$ specifies that its children are played one child at a time, but does not impose any order. The $\langle \text{par} \rangle$ plays all children elements as a group, allowing parallel play out. For example, the SMIL specification $\langle \text{par} \rangle \text{ video src=camera1 } \langle \text{audio src=microphone1} \rangle / \langle \text{par} \rangle$ specify that media sources camera1 and microphone1 are played in parallel.

In SMIL, the time period that a media clip is played out is referred to as its *active duration*. For parallel play to be meaningful, both sources must have equal active durations. When clips do not have equal active durations, SMIL provides many constructs to equate them. Some examples are *begin* (allows to begin components after a given amount of time), *dur* (controls the duration), and *end* (specifies the ending time of the component with respect to the whole construct). In this paper we assume that children of $\langle \text{par} \rangle$ have active durations.

4 Object Identity in SMIL

For XML formatted textual documents the *protection objects* are nodes of the XML tree. This may be acceptable for some forms of multimedia, such as movies [KW02]. But as shown in section 1 using Fig. 1, this is problematic for multimedia in general. We therefore define the SMIL normal form in Definition 1.

Definition 1 (SMIL Normal Form) *We say that a SMIL specification(s) is in the SMIL Normal Form (smilNF) if it is of the following form $\langle \text{seq} \rangle \langle \text{par} \rangle C_{1,1}(s) C_{1,2}(s) C_{1,3}(s) \dots C_{1,n}(s) \langle / \text{par} \rangle \dots \langle \text{par} \rangle C_{m,1}(s) C_{1,2}(s) C_{1,3}(s) \dots C_{m,n}(s) \langle / \text{par} \rangle \langle / \text{seq} \rangle$ where $C_{i,j}$ are audio or video media intervals.*

Figure 2 shows a more general representation of SMIL objects. In Representation 4, there are 4 sequentially arranged audio or video frames which in turn are time-sliced into three intervals. The boxes represented by $A_1, A_2 \dots D_2, D_3$ could be either a audio or video frame. The right hand side of the representation shows how it is represented in the normal form according to Definition 1.

As stated, a sub object of a SMIL object does not have to be sub tree of one of its syntactic representation. In Representation 5, the sub-object we consider is shown by the enclosed area, and its normal form tree is shown on the right hand side.

4.1 Security Paradigms and Access Control Rules

In order to specify security policies the *subject* and the *protection object* need to be unambiguously identifiable. The subject may be granted an access permission in DAC, but in MLS and RBAC such granting is indirect and has to satisfy some constraints, usually expressed in the form of rules. This section formally defines the security paradigms we use and the associated constraints that are used to construct the access control lists.

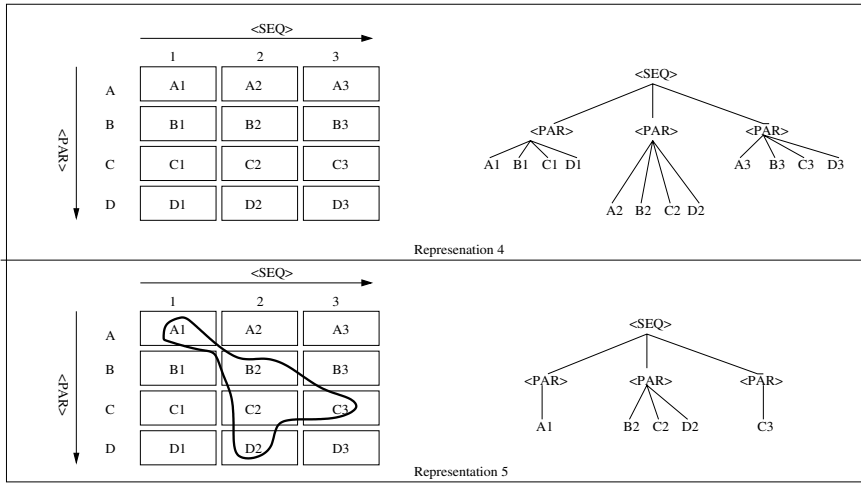


Fig. 2. A Generalized Representation based on the Normal Form

DAC (Discretionary Access Control). Discretionary Access Control defines access permissions a based on subjects s and objects o . Such a permission can be expressed by constructing an access control matrix containing appropriate triples (s, o, a) .

RBAC (Role Based Access Control). The simplest Role-Based Access Control models has three entities roles, users, privileges, and two associations, subject-to-role and role-to-privilege assignments among them. A subject may activate any authorized roles, and by doing so obtains all privileges assigned to the activated role.

For each subject s let the set of active roles be given by $ActR(s)$, and $AuthR(s)$ be the set of roles permitted to be invoked by s . Then, the restriction that a user may activate only authorized roles can be stated as $ActR(s) \subseteq AuthR(s)$.

Privileges (access permissions) associated for each role are based on objects defined in the rbacNF. That is, a given specification S in rbacNF is organized in a manner that all objects permitted to a role R_i are represented together. Then, we can define the access permissions of each role r as $rToPer(r_i)$, where $rToPer(r_i)$ consists (object, action) pairs. Then (s, o, a) belongs to the access control matrix iff $ActR(s) \subseteq AuthR(s) \wedge \exists r \in ActR(s) (o, a) \in rToPer(r)$.

MLS (Multi Level Security). In Multi Level Security each access permission is guided by the security clearance of the subject and the security classification of the accessed object. Security labels form a lattice structure with the dominance relation among the labels. Information flow between the security labels is controlled based on the security objectives. In this paper we allow information flow from low security objects to high security objects, that is, from a dominated

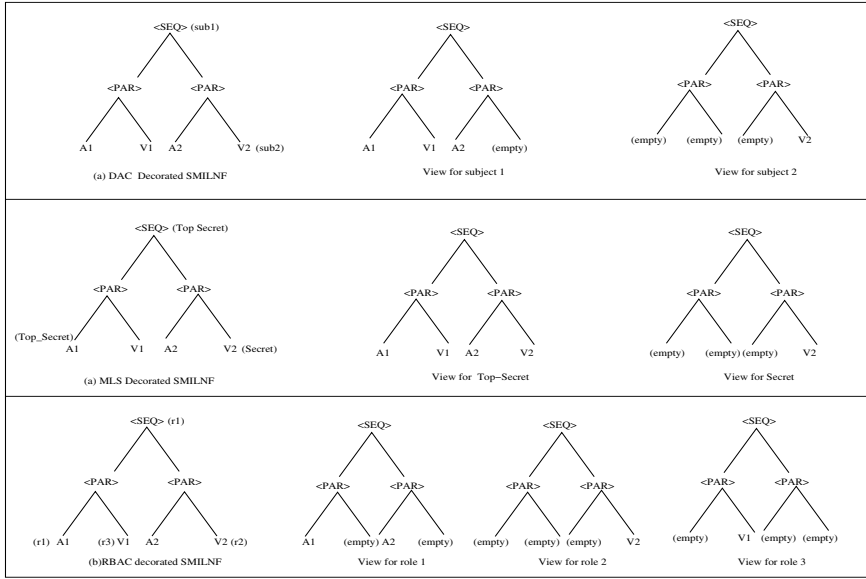


Fig. 3. Reduction to dacNF, mlsNF and rbacNF

object to a dominating object. Assuming that our access permissions are “read” permissions, it means that a subject is allowed to access an object only if the subject’s security clearance dominates the security classification of the object.

Let $Class(s)$ denote the classification of subject s . L denotes the lattice structure and binary relation $dominates(l_1, l_2)$, $l_1, l_2 \in L$ denotes that label l_1 dominates label l_2 . To generate all labels dominated by the security classification of a subject ($Class(s)$), we generate the transitive closure of dominance relation as follows:

1. Let $Dominated(s) = Class(s)$
2. For all pairs $dominates(l_i, l_j)$, where l_i in $Dominated(s)$, $Dominated(s) = Dominated(s) \cup l_j$

To permit accesses for a subject to objects in mlsNF, we use the set $Dominated$ to determine the appropriate data items. That is,

$\forall s$, if $Class(s)$ and $\{l_{i_1}, \dots, l_{i_n}\} \in Dominated(s)$ and $o \in Cl_{i_k}$ $k = 1, \dots, n$ then (s, o, a) .

That is, a subject is granted the access a to an object o if the security clearance of the subject dominates the security classification of the object. Hence MLS could be stated as an (s,o,a) triple. In effect, the generalized access control rule in all three domains could be declared as a (s,o,a) triple.

5 Secure Normal Forms

As briefly described DAC, MLS and RBAC security policies can be reduced to (s,o,a) triples. However in RBAC the permissions are assigned primarily to roles and subject's permission (that is (o,a) pairs) could be derived depending on a subjects active roles. Similarly, in MLS permissions are assigned to security levels, and depending on the clearance of the subjects, subject's permission (that is (o,a) pairs) could be derived. Therefore, we allow SMIL documents in smilNF to be decorated to subjects, security levels and roles respectively. Then the final authorization triples (s,o,a) triples can be derived using appropriate rules.

The security decoration on the *protection object* is defined on the normal form. We allow any node of a SMIL tree in smilNF to be decorated as shown in the Fig. 3. Given any such decoration, we can compute a view that is permitted for each subject, security level or a role. They are referred to as *security normal forms*. Security normal forms are formally defined in Definitions 2, 3, 4.

5.1 Normal Form for DAC

The DAC normal form is a parallel composition of permitted segments. The smilNF specification is decorated with the DAC metadata, and upon reduction, would group all permitted segments of a particular subject under a single $\langle \text{par} \rangle$ construct. Each of these $\langle \text{par} \rangle$ construct is the *view* of the associated subject.

Definition 2 (DAC Normal Form) We say that a smilNF specification ($\tilde{s}$) is in the DAC Normal Form (dacNF) if it is of the form $\langle \text{seq} \rangle \langle \text{par} \rangle C_1(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_2(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_3(\tilde{s}) \dots C_n(\tilde{s}) \langle / \text{par} \rangle \langle / \text{seq} \rangle$ where $C_1, C_2, C_3 \dots C_n$ are media intervals permitted to be accessible to security level.

5.2 Normal Form for MLS

Definition 3 (MLS Normal Form) We say that a smilNF specification ($\tilde{s}$) is in the mlsNF (MLS Normal Form) if it is of the form $\langle \text{seq} \rangle \langle \text{par} \rangle C_t(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_s(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_u(\tilde{s}) \langle / \text{par} \rangle \langle / \text{seq} \rangle$ where all Security classifications in $C_t(\tilde{s})$, $C_s(\tilde{s})$, $C_u(\tilde{s})$ are respectively Top-Secret, Secret and Unclassified.

As stated in Definition 3, a Normal Form in mlsNF is one that is a parallel composition of at most three documents, where each document belongs to one security class, that are said to be the views corresponding to the respective security classes.

5.3 Normal Form for RBAC

Definition 4 (RBAC Normal Form) We say that a smilNF specification ($\tilde{s}$) is in the rbacNF (RBAC Normal Form) if it is of the form $\langle \text{seq} \rangle \langle \text{par} \rangle C_{r_1}(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_{r_2}(\tilde{s}) \langle / \text{par} \rangle \langle \text{par} \rangle C_{r_3}(\tilde{s}) \dots C_{r_n}(\tilde{s}) \langle / \text{par} \rangle \langle / \text{seq} \rangle$ where the Role attributes in $C_{r_1}(\tilde{s})$, $C_{r_2}(\tilde{s})$, $C_{r_3}(\tilde{s}) \dots C_{r_n}$ are respectively $role_1, role_2, role_3 \dots role_n$.

As stated in Definition 4, a Normal Form in rbacNF is one that is parallel composition of at one or more role specifications, where each specification belongs to a particular role assignment, and is said to be the view corresponding to the assigned role.

5.4 Conversion into Secure Normal Forms

In Fig. 3 we have three examples of decorated smilNF. The security classification could be done at three levels the primary time container, the nested time container and at the frame level. The Fig. 3 shows the schematic reduction after applying the algorithms listed in previous sections. In our DAC example subject sub_1 is permitted access to the whole tree, where as subject sub_2 is granted access only to video frame V_2 . The reduction uses the $\langle \text{empty} \rangle$ to denote an element that is disallowed. The views corresponding to sub_1 and sub_2 that when combined form the dacNF after the application of the algorithm is shown on the right hand side. The first composition denotes the *view* of subject sub_1 and the second composition the *view* of subject sub_2 . In the MLS example the $\langle \text{par} \rangle$ is classified as Top-Secret and audio frame A_1 is also classified as Top-Secret. The video frame V_2 is classified as secret. The resulting mlsNF is a parallel composition of two security classifications, and the Top-Secret (higher classification) is allowed access to the Secret (lower) classification by the virtue its position in the classification hierarchy. Similarly a RBAC decorated smilNF with roles $r_1, r_2, r_3 \dots r_n$ would produce an rbacNF with the "n" number of roles.

The algorithm for conversion from smilNF to mlsNF would generate output that would have atmost three parallel compositions each corresponding to a unique security level. The MLS paradigm has an unique property which allows subjects with a higher classification access to the view of the lower classified subjects. This algorithm takes this property into consideration when generating smilNF. The algorithm for RBAC would yield views for each role and the algorithm for DAC would be similar.

During the rewrite, some of the nodes are represented as $\langle \text{empty} \rangle$. This representation is used to establish an audio or video *silence* in the payout. When grouping elements that satisfy a particular access control rule, there is a need to eliminate those that do not qualify. Normally, a silent audio segment or a blank video segment are used to during payout to maintain continuity without losing synchronization.

6 Metastructure

Metadata is needed for specifying access control policies for multimedia because the current specification of SMIL [Aya01] does not have constructs for security and minimal constructs for QoS. The SMIL metamodule [Mic01] claims that RDF could be used to declare metadata to be used within a SMIL document, but does not provide sufficient detail on how to effectively use RDF to state our needs. The RDF[KC03] and RDFS[BG03] enable defining metadata but

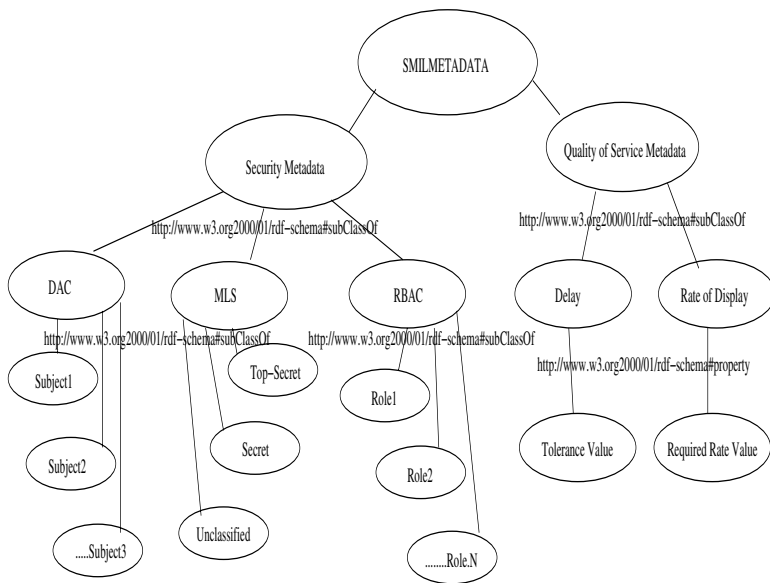


Fig. 4. Class Hierarchy of the Metastructure

not the interpretation or anticipated meaning applicable to multimedia. Consequently, we design a structure for metadata to enforce security related to various paradigms.

6.1 Resource Description Framework

RDF (Resource Description framework) is a language for representing information about resources that can be identified on the web. The URI (Uniform Resource Identifiers) with optional fragment identifiers are used to describe subjects objects and predicates in statements, and relationships between URI-identifiable entities. This representation primarily uses RDF/XML, but because our focus is synchronized multimedia the representation is in RDF/SMIL. In this section we describe the a RDF metastructure for secure multimedia using RDF-Schema. Our vocabulary is defined in a namespace identified by the URI reference <http://www.w3.org/2000/01/rdfschema/#> . In the following structure the prefix *rdfs* is used to refer to that namespace.

As stated using the RDF/XML [MM03] we define the *xmlns*(XML namespace) for the metadata and call it *smilmetadata*. We refer to *smilmetadata* in order to use any metadata we define. The description *smilmetadata:MLS* is useful in identifying permissible media elements within a SMIL-formatted document when our security paradigm is MLS (Multi-Level-Security).

Figure 4 represents the class hierarchy of the metadata we define in RDF for specifying security and QoS in a SMIL formatted multimedia document. Figure 4

represents those components necessary to represent security and QoS parameters chosen for this study.

The metastructure we define is based on a schema and represents meta-data for our chosen security and QoS parameters. In the context of security we need to define metadata to effectively represent the security paradigm with respect to DAC, MLS and RBAC. The MLS class, consists of Top-Secret, Secret and Unclassified as sub-classes. RBAC and DAC have subjects and roles defines as sub-classes. Our QoS metric consists of two parameters: **delay** and **rate of display** under the class Run-time-QoS. These attributes take values **toleranceValue** and **requiredRateValue** respectively. The namespace for the metastructure is <http://svp.gmu.edu/smil-ns#> and is referred to as *smilmetadata*.

The **subClassOf** in RDF Schema is a special subset/set relationship between two classes. In the metastructure, Top-Secret, Secret and Unclassified are sub-classes of Class: MLS. The **rdf:subClassOf** property is transitive, implying that resources that are instances of **subClass** are implicit instances of the Class. The **rdf:domain** and **rdf:range** attributes available in RDF are used to define the scope of the members of a container with respect to a property of a class.

1. **MLS** is used to define the security level of a particular media element. The actual levels of Security [Top-Secret, Secret, Unclassified] are sub-classes of the class MLS.
2. **RBAC** defines the role of the current role assigned to the object.
3. **Run-time-QoS** regulates different parameters for maintaining good service during the delivery of the media.

As specified by W3C interpretation of the metadata is entirely the responsibility of the application that uses them.

The security metadata used to decorate a SMIL specification is supposed to reflect the security paradigm used. The DAC is enforced through subjects and the MLS through security classifications, Top-Secret, Secret and Unclassified and the RBAC through role attributes. For e.g $\langle \textit{smilmetadata} : \textit{MLS} \rangle$ enclosing $\langle \textit{smilmetadata} : \textit{Top-Secret} \rangle$ means we refer to the Top-Secret **rdf:subclass** of the *smilmetadata*: MLS class to define the security attribute of an element. The metastructure needs to describe QoS parameters. Among a choice of many, we only consider *delay* and *rateOfDisplay* as the minimum negotiable application level parameters and are described as stated above. The **toleranceValue** and **requiredRateValue** are properties used to specify the requirements to the clients during delivery of the media.

7 Operational Semantics

Our metastructure can be used by a multimedia client that seeks to obtain SMIL documents with proposed RDF decorations. Our client must use an RDF based query system for this purpose to generate views for DAC, MLS and RBAC. The RDF Query [MS98] uses a declarative syntax for selecting RDF resources

that meet specified criteria. For example, for RBAC retrieval, we show how to construct a RDF query to retrieve the view for a given role. Similarly, we show an example query to retrieve all objects corresponding to particular security classification.

An RDF-interpreter is necessary to understand and assemble a SMIL view from a RDF decorated SMIL document that is to be interpreted by a SMIL player at the client. Although we do not provide such an interpreter, our client need to have two interacting interpreters, where the SMIL-Interpreter calls the RDF-interpreter to interpret RDF decorations.

As stated in Section 4.1, all DAC, MLS and RBAC can be reduced to the access control rule could be stated as a simple (s: subject, o: object, a:access). Therefore the access control rule is defined as a 4 tuple (c,o,d,a) where C is a condition expressed in RDF Query, o is the security object(Normal Form), d is the decision to grant or deny and a is the action to be performed when this rule is activated.

An example of RDF Query [MS98] for the RBAC and MLS security paradigms are discussed in Section 7.1 and 7.2. The conditions use SQL keywords such as *select*, *from* etc. Complex and nested queries could be formulated with the use of boolean expressions.

7.1 An RBAC Query

This query represented below retrieves the view pertaining to a single role (*role₁*) from the rbacNF. The scope of the RBAC query is the RBAC Normal form. The structure of rbacNF guarantees that media components associated with the particular role is grouped together, and the retrieval could be based on the metadata used to define the particular role assignment. The RBAC query in section 7.1 would **select** components associated with *smilmetadata: role₁* **from** the specified URI for the location of the rbacNF.

```
< rdfq:rdfquery >
< rdfq:From eachResource="http://svp.gmu.edu/AV/smil-ns # rbacNF "/" >
  < rdfq:Select >
    < rdfq:Property name=" role1 "/" >
      < /rdfq:Select >
  < /rdfq:From >
< /rdfq:rdfquery >
```

The query below retrieves the view pertaining to a specified security classification within a MLS Normal Form. The scope of the MLS query is the mlsNF represented by the appropriate URI. The MLS query in section 7.2 would **select** components associated with *smilmetadata :Top-Secret* **from** the specified URI that denotes the location of the mlsNF.

7.2 MLS Query

```

< rdfq:rdfquery >
< rdfq:From eachResource="http://svp.gmu.edu/AV/smil-ns #mlsNF"/ >
  < rdfq:Select >
    < rdf:ID > Top-Secret < /rdf:ID >
    < /rdfq:Select >
  < /rdfq:From >
< /rdfq:rdfquery >

```

A The run-time algorithm describes the retrieval of a secure SMIL document. During the first stage, the algorithm negotiates the QoS parameters. A failure of available QoS would result in the termination of the media transfer. Once the query answer is obtained, the access control policy is evaluated. If access is granted the associated action is initiated. Views could be encrypted to enforce integrity and unwanted stream acquisition and guarantee unforgability. Several encryption techniques can be used, such as the ones suggested in [KWJ03,KW02].

8 Conclusions

We showed that syntactic trees used in textual XML documents to specify access control policies are insufficient to specify access control policies for SMIL formatted multimedia documents. As a solution, we proposed that SMIL documents to be translated to a normal form similar to the DNF representation of propositional formulas.

Having resolved the issues of objects and their identity in SMIL, we presented a RDF metastructure to specify access control policies for multimedia documents. We have shown via examples the applicability of the structure for DAC, MAC, and RBAC. Our security normal forms are similar to secure views computed for XML and other textual documents. We showed a straw-man's design of a run-time that uses RDF and SMIL queries to securely retrieve documents decorated as specified by us.

Results presented here only consider limited aspects of security models with a fragment of SMIL syntax. Our ongoing work addresses these limitations and provide comprehensive security models. In addition, we are incorporating advanced Semantic Web technologies, like DAML+OIL [CHH01], OWL [DC03] and RuleML [BTW01].

References

- [Aya01] Jeff Ayars. *Synchronized Multimedia Integration Language*. W3C Recommendation, 2001. <http://www.w3.org/TR/2001/REC-smil20-20010807>.
- [BG03] Dan Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0:RDF Schema*. W3C Working Draft, January 23 2003. <http://www.w3.org/TR/2003/WD-rdf-schema-20030123>.

- [BHA02] Elisa Bertino, Moustafa Hammad, Walid Aref, and Ahmed Elmagarmid. An access control model for video database systems. In *Conferece on Information and Knowledge Management*, 2002.
- [BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design rationale of ruleml: A markup language for semantic web rules. In *SWWS, Stanford*, 2001.
- [CHH01] Dan Connolly, Frank Harmelen, and Ian Horrocks. *DAML+OIL Reference Description*. W3C Note, 2001. <http://www.w3.org/TR/daml+oil-reference>.
- [DC03] Mike Dean and Dan Connolly. *OWL Web Ontology Language Overview*, 31st March 2003.
- [DdV03] Ernesto Damiani and Sabrina De Capitani di Vimercati. Securing xml based multimedia content. In *18th IFIP International Information Security Conference*, 2003.
- [GNY⁺01] Xiaohui Gu, Klara Nahrstedt, Wanghong Yuan, Duangdao Wichadakul, and Dongyan Xu. An xml-based quality of service enabling language for the web, 2001.
- [Hay03] Patrick Hayes. *RDF Semantics*. W3C Working Draft, January 23 2003. <http://www.w3.org/TR/2003/WD-rdf-mt-20030123>.
- [KC03] Graham Klyne and Jeremy Carroll. *Resource Description Framework(RDF) Concepts and Abstract Syntax*. W3C Working Draft, January 23 2003. <http://www.w3.org/TR/2003/WD-rdf-concepts-20030123>.
- [KFW03] Naren Kodali, Csilla Farkas, and Duminda Wijesekera. Enforcing integrity in multimedia surveillance. In *IFIP 11.5 Working Conference on Integrity and Internal Control in Information Systems*, 2003.
- [KW02] Naren Kodali and Duminda Wijesekera. Regulating access to smil formatted pay-per-view movies. In *2002 ACM Workshop on XML Security*, 2002.
- [KWJ03] Naren Kodali, Duminda Wijesekera, and J.B.Michael. Sputers: A secure traffic surveillance and emergency response architecture. In *submission to the Journal of Intelligent Transportation Systems*, 2003.
- [Mic01] Thierry Michel. *The SMIL 2.0 MetaInformation Module*. W3C Recommendation, 2001. <http://www.w3.org/TR/2003/WD-rdf-mt-20030123>.
- [MM03] Frank Manola and Eric Miller. *RDF Primer*. W3C Working Draft, January 23 2003. <http://www.w3.org/TR/2003/WD-rdf-primer-20030123>.
- [MS98] Ashok Malhotra and Neel Sundaresan. *RDF Query Specification*. W3C Specification, December 03 1998. <http://www.w3.org/TR/2003/WD-rdf-primer-20030123>.
- [SFK00] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NISI model for role-based access control: Towards a unified standard. In *ACM RBAC 2000*, pages 47–64, 2000.
- [SS96] Ravi Sandhu and Pierangela Samarati. Access control: Principles and practices. *IEEE Communications*, 29(2):38–47, 1996.
- [W3C03] *World-Wide-Web Consortium*, 31st July 2003.
- [WS96] Duminda Wijesekera and Jaideep Srivastava. Quality of service (qos) metrics for continuous media. *Multimedia Tools and Applications*, 3(2):127–166, 1996.

Managing Identities via Interactions between Ontologies

Paolo Ceravolo

Department of Information Technology
University of Milan
Via Bramante, 65 - 26013 Crema (Italy)
`pceravolo@crema.unimi.it`

Abstract. In this paper, we describe how an *identity management* system can be based on *user ontologies* in order to deal with complex attributes that are needed to model user interests or relationships. The problem of computing efficient bindings between ontology based metadata and XML based standards like SAML is also discussed.

Keywords: Ontology, Identity Management, Multiple Identities.

1 Introduction

Profiling users is a widely used technique for customizing e-business services, increasing their effectiveness and usability. On the other hand, profiling has given rise to privacy concerns due to unauthorized cross-site profiling of users' tastes and habits. In order to deal with such concerns, some sort of control must be established on storage and management of profile information. Many approaches to this problem propose to give control of profile information back to users, leaving them the last word on whether the data about themselves they disclose during transactions should be made available for exchanging or sharing among different services. While attractive in principle, however, this solution proved to be hard to implement: indeed, it is difficult to prevent sites holding data about their users from sharing or merging them to obtain complete profiles. Here, we adopt a more *human-centered* [12] perspective, dealing with *digital identities* rather than user profiles. Informally speaking, a digital identity is a set of attributes describing (an aspect of) a user. When a user performs on the Net a set of e-business transactions involving multiple services, *digital identity management* is the act of deciding which information will be made available to which remote service. Identity management can prevent unauthorized profiling by letting a single user be known under different nicknames or pseudonyms in different contexts. *Identity management systems* are software agents that help people in defining different identities, associating personal data to them, and deciding which information should be disclosed when interacting with different applications. Some projects that seem to follow similar ideas have been launched in the last few years, namely Microsoft Passport, Novell DigitalMe, the Liberty

Alliance project and XNS. However, most of these solutions follow a centralized approach and do not give users full control over their identity information. Also, it is important to remark that these projects do not provide a rich representation of user-related information comparable to the one stored in conventional user profiles. In this paper, we follow a different line of research, describing how an identity management system could be based on *user ontologies* in order to deal with complex attributes that are needed to model user interests or relationships. The paper is structured as follows: Section 2 outlines our approach, while Section 4 introduces via some examples the notion of bindings between ontology-based metadata and XML based standards like SAML. Finally, Section 5 draws the conclusions and mentions some directions for our future research.

2 Managing Multiple Identities via Ontology

The issue of digital identity management is being increasingly discussed both within the security research community and among officials of regulatory bodies. The reason for this high level of interest is twofold:

1. Subjects must store and manage an increasing amount of information about properties they have or credentials they hold in order to carry out online transactions. successfully.
2. Information on users should not be disclosed carelessly: at each transaction step, a subject should reveal only the minimum amount of it that is strictly needed to carry on the transaction.

Current e-business infrastructure does not support controlled disclosure or negotiation of subjects' attributes or credentials, and therefore can give rise to privacy leaks or uninformed choices on the part of the users [2]. In this paper, we advocate the use of advanced knowledge representation techniques, such as schemata encoding ontologies, as the basic layer of an identity management system. Ontologies have been introduced since long as fundamental tools for representing domain and task knowledge via networks of concepts, and have been often used to represent the semantics of online transactions. For instance, the DAML-S proposal is an ontology syntax explicitly aimed at integrating ontologies and Web Services in a single framework. Besides allowing for efficient management of information about user properties and credentials, user ontologies support unambiguous specification of access rules [9]. In this paper, for the sake of clarity, we shall use the widespread language OWL for our examples. Also, we shall rely on our informal definition of an ontology as a network of concepts rather than attempting to give a more formal one. A complete survey of standard languages for ontology representation can be found in [1].¹

¹ For the sake of conciseness, here we do not try to give a description of ontology-related metadata standards. The essential references can be found at the following sites: RDF - RDFS <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/> and <http://www.w3.org/TR/rdf-schema/>, OWL <http://www.w3.org/TR/owl-ref/>, SAML www.oasis-open.org/committees/security/.

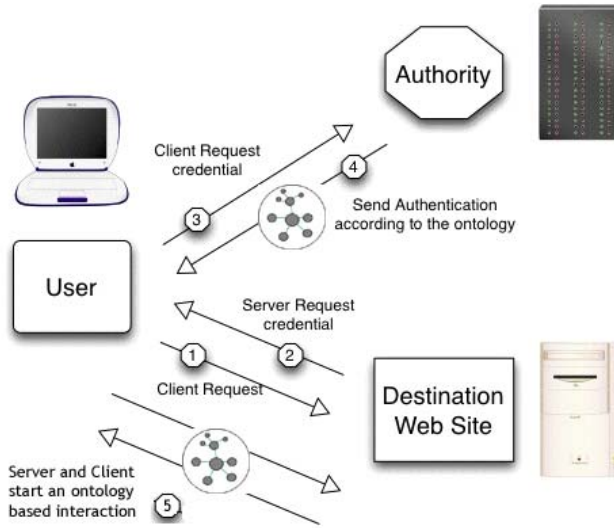


Fig. 1. The role of the ontology in the transaction architecture

2.1 A Usage Scenario

We are now ready to present a basic scenario featuring an ontology-based identity management system. Figure 1 summarizes our scenario. Let us suppose that a client needs to invoke a remote service. The service will require some credentials in order to grant access, specifying some information on the client

1. Instead of sending all the available information about itself to the server and letting the remote party sort out which information is relevant to the request, the client consults the identity management system (called **Authority** in Fig. 1) in charge of preserving its privacy.
2. The identity management system, in turn, uses a *personal identity ontology* in order to select the concept expressing “right” amount of information that the client needs to disclose.
3. An instance of this concept is then sent to the server².
4. Concept selection may consist in an educated guess on the part of the system, or be negotiated via a dialogue with the server as shown in Fig. 1.

3 Interactive Configuration of Identity Attributes

We shall now describe a sample personal identity ontology, designed in such a way to express the different variations of the “user identity” concept when accessing

² Note that here we are not concerned with the values of the instance being the ones the server expects to grant access, but only with the fact that the instance contains the exact amount of information that the server needs in order to make a decision

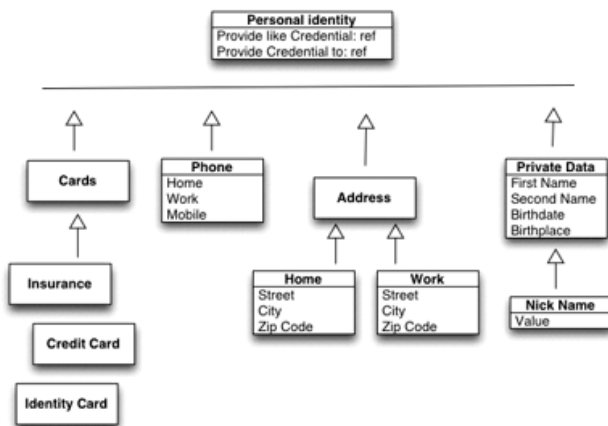


Fig. 2. personal-identity

remote services. Our ontology is written in the OWL standard language, whose expressive power allows for representing *logical constraints*.

All classes in the identity ontology of Fig. 3 have two relations, each connecting them to an external ontology. Namely, relation **Provide-as-Credential** ranges into a **Credentials** ontology (Fig. 3). The target ontology defines for each attribute of the “user identity” concepts the credentials needed to certify it. In turn, relation **Provide-Credential-to** may range into a **Providers** ontology specifying the different types of authorities that can provide such credentials to users.

4 Bindings

There is of course a potential mismatch between the instance of a concept belonging to the **Credentials** ontology generated by the identity management system, which has been defined by an external Provider, and the format of the actual identification data the remote service expects. This problem can be solved by defining suitable bindings from the **Credentials** ontology concepts to XML-based standards used for providing subject information. Here we shall focus on SAML, the widely adopted standard proposed by OASIS³. The most intuitive way to create a binding from ontology-based metadata to SAML is using the information contained in the ontology concept’s instance to fill in the SAML request to be sent to the remote service⁴. To this end, the identity management

³ A complete introduction to SAML can be found at www.oasis-open.org/SAML.

⁴ In practice, it might prove to be more implementation-friendly to fully decouple the ontology-based metadata from SAML by defining a lower level ontology specifying the SAML-to-Credentials mapping [4], and then using instances of those lower-level concepts to generate SAML. We decided to refrain from discussing this technicality in this paper.

```

<owl:Class rdf:ID="Personal-identity">
  <rdf:subClassOf rdf:resource="#Resource"/>
</owl:Class>

<owl:objectProperty rdf:ID="Provide-as-Credential">
<rdfs:domain rdf:resource="#Personal-Identity"/>
<rdfs:range rdf:resource="#Credential"/>
</owl:objectProperty>

<owl:objectProperty rdf:ID="Provide-Credential-to">
<rdfs:domain rdf:resource="#Personal-Identity"/>
<rdfs:range rdf:resource="#Provider"/>
</owl:objectProperty>

<owl:Class rdf:ID="Address">
  <rdf:subClassOf rdf:resource="#Personal-Identity"/>
</owl:Class> <owl:Class rdf:ID="Home">
  <rdf:subClassOf rdf:resource="#Address"/>
</owl:Class> <owl:Class rdf:ID="Work">
  <rdf:subClassOf rdf:resource="#Address"/>
</owl:Class> <owl:Class rdf:ID="Cards">
  <rdf:subClassOf rdf:resource="#Personal-Identity"/>
</owl:Class>

<owl:Class rdf:ID="Card">
  <owl:unionOf rdf:parserType="Collection">
    <owl:Class rdf:about="#Insurance-Card"/>
  <owl:Class rdf:about="#Credit-Card"/>
    <owl:Class rdf:about="#Identity-Card"/>
  </owl:unionOf>
</owl:Class>

```

Fig. 3. The Personal Identity Ontology

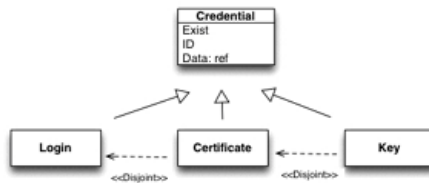


Fig. 4. The Credential Ontology

system must interact with the local directory service holding all available subject data in order to retrieve the needed information. Here is a sample of a **Credit Card** instance containing a credential.

```

<owl:Class rdf:ID="Credential">
  <rdf:subClassOf rdf:resource="#Resource"/>
</owl:Class>

<owl:objectProperty rdf:ID="Exist">
  <rdfs:domain rdf:resource="#Credential"/>
  <rdfs:range rdf:resource="#saml:Request"/>
</owl:objectProperty>

<owl:objectProperty rdf:ID="ID">
  <rdfs:domain rdf:resource="#Credential"/>
  <rdfs:range rdf:resource="#saml:Request"/>
</owl:objectProperty>

<owl:objectProperty rdf:ID="Data">
  <rdfs:domain rdf:resource="#Credential"/>
  <rdfs:range rdf:resource="#saml:Request"/>
</owl:objectProperty>

<owl:Class rdf:ID="Login">
  <owl:disjointWith rdf:resource="#Certificate"/>
  <owl:disjointWith rdf:resource="#Key"/>
  <owl:subClassOf rdf:resource="#Credential"/>
  <owl:subClassOf>
</owl:Class>

```

Fig. 5. The Credentials Ontology

```

<a:Credit-Card rdf:about="http://website.org/instances/model#Instance-001">
  <a:Provide-as-Credential
    rdf:resource="http://websiteB.org/instances/model#Instance-002"/>
  <a:Provide-Credential-to
    rdf:resource="http://websiteB.org/instances/model#Instance-003"/>
</a:Credit-Card>

<a:Certificate rdf:about="http://websiteB.org/instances/model#Instance-004">
  <a:ID="<samlp:Request.." />
  <a:Data="<samlp:Request.." />
</a:Credit-Card>

```

The SAML attributes corresponding to the Credit Card concept's ones follow:

```

<samlp:Request>
<samlp:AttributeQuery>
<saml:Subject></saml:Subject>
  <saml:AttributeDesignator
    AttributeName="Credit-Card"
    AttributeNamespace="http://websiteA.com">
    <saml:AttributeDesignator

```

```

    AttributeName="amount"
    AttributeNamespace="http://websiteA.com"/>

    <saml:AttributeDesignator
      AttributeName="number"
      AttributeNamespace="http://websiteA.com"/>

    <saml:AttributeDesignator
      AttributeName="owner-name"
      AttributeNamespace="http://websiteA.com"/>

    <saml:AttributeDesignator
      AttributeName="credit-limit"
      AttributeNamespace="http://websiteA.com"/>
  </saml:AttributeDesignator>
</samlp:AttributeQuery>
</samlp:Request>

```

Now, the transformation can be computed in order to obtain SAML markup. The resulting SAML assertion is shown below:

```

<saml:Assertion>
  <saml:AttributeStatement>
    <saml:Subject></saml:Subject>
    <saml:Attribute
      AttributeName="Credit-Card"
      AttributeNamespace="http://websiteB.com">
    <saml:Attribute AttributeName="CreditLimit"
      AttributeNamespace="http://smithco.com">
    <saml:AttributeValue xsi:type="my:type">
    <my:amount currency="Euro">500.00</my:amount>
    </saml:AttributeValue>
    ...
  </saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>

```

The binding from the instance of the **Credentials** ontology concept to the SAML assertion can be formally represented as a transformation on *XML graphs*.^[4] Figure 6 shows the **Credit Card** credential and its SAML binding as a graph transformation.

4.1 Efficient Binding Computation: The Issues

In order to be feasible for practical applications, metadata bindings need to be computationally efficient. When structure transformations are involved, access control research developed some high-performance XML transformation techniques which could be applied to bindings^[13]. Structure transformation is not

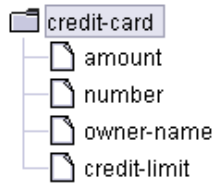


Fig. 6. A graph representing the SAML query

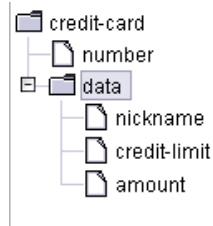


Fig. 7. A graph representing the SAML response

all that is needed for efficient binding computation. Indeed, in our example we need to perform some content as well as structure adaptation: the server requires a **Owner-Identity** tag, e.g. to check whether the payment was made by the right person in cases where payment on behalf of others are not allowed without proof of consensus. Ontology-based metadata in Fig.6 only provide a nickname. However, suitable content can be easily synthesized. Since our ontology contains references to external certificates, provisions can be taken in order to check them, e.g. delegating the task to an auxiliary authority. This delegation will provide the desired identity control. In other words, matching between the two involved schemata (the source and target of the binding) at the content level is obtained via fast transformation of the structure and cooperative computation of the content. The efficiency of the latter process is an open issue currently being investigated.

5 Conclusions and Further Developments

In this paper, we introduced the notion of ontology based metadata for identity management and gave a preliminary discussion of computing bindings from such metadata to XML related standards like SAML. We are currently investigating the application of fuzzy structure matching [4] to the problem of inducing identity ontology organization from SAML sample data.

Acknowledgements. The author wishes to thank Ernesto Damiani and Pierangela Samarati for their contribution to the main ideas in this paper, as well as for their continuous guidance and valuable advice.

References

1. Denker, G.: Towards Security in DAML. Internal Report, *SRI International, Menlo Park, CA 94025*, [2002].
2. Clauss, S.: Identity management and its support of multilateral security, *Computer Networks (37)*, 205–219. [2001].
3. Golbeck J., Parsia B., Hendler J.: Trust networks on the semantic web, *In Proceedings of Cooperative Intelligent Agents*, 5 [2003].
4. Damiani E., Tanca L.: Blind queries to XML data, *SEBD 79–93* [2000].
5. Damiani E., Tanca L., Arcelli Fontana F.: Fuzzy xml queries via context-based choice of aggregation, *Kybernetika, vol. 36, num. 6, pag 79–93* [2000].
6. Maedche, A., Staab S.: Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2), [2001].
7. Mena, E., Illarramendi, A.: Ontology-based Query Processing for Global Information Systems. *Kluwer Academic Publishers*, Boston [2001].
8. Davies, J., Fensel, D., Van Harmelen, F. (eds.): Towards the Semantic Web: Ontology-Driven Knowledge Management, *J. Wiley* [2002].
9. Bonatti, P., Samarati P.: Regulating Service Access and Information Release on the Web, *in Proc. Seventh ACM Conference on Computer and Communications Security, Athens, Greece, November 1–4*, [2000].
10. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, Vol.5, pp. 199–220 [1993].
11. Gruber, T. R.: Toward principles for the design of Ontologies used for Knowledge Sharing. *Technical Report KSL 93–04*, Knowledge Systems Laboratory, Stanford University [1993].
12. Khosla, R., Damiani, E., Grosky, W.: Human-centered E-business, *Kluwer Academic Publisher*, [2003].
13. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati P.: Securing XML Documents, *in Proc. WWW Conference, Amsterdam, The Netherlands* [2000].

Implementing an Access Control System for SVG Documents

E. Fernández-Medina¹, G. Ruiz¹, and S. De Capitani di Vimercati²

¹ Escuela Superior de Informática
Universidad de Castilla-La Mancha
13071, Ciudad Real - Spain
`Eduardo.FdezMedina@uclm.es`

`gruiz@proyectos.inf-cr.uclm.es`
² Dip. di Tecnologie dell'Informazione
Università degli Studi di Milano
26013 Crema - Italy
`decapita@dti.unimi.it`

Abstract. In this paper, we present an access control system that can be used for controlling access to SVG documents. The first part of this paper briefly describes the access control model on which the system is based. The second part of this paper presents the design and implementation of the system.

1 Introduction

An increasing amount of multimedia information transmitted over the Internet is in the form of vector image data, encoded by means of new XML-based standards such as the World Wide Web Consortium's *Scalable Vector Graphics* (SVG) [10], which allows for the definition of two dimensional vector graphics (specifically vector graphic shapes, images, and text) for storage and distribution on the Web. The SVG standard can be used in different applications: technical plans, organizational charts and diagrams, as well as medical images used in diagnosis and research, to name a few. In the security area, while controlling access to text-based documents has been the focus of many research activities [9], raster graphic information has been seldom considered, mainly because of its monolithic internal structure. However, XML-based vector images present new and challenging *feature protection* problems, related to fine-grained access control to their internal structure. We have then defined a novel approach to fine-grained feature protection of Scalable Vector Graphics (SVG) data [1]. The proposed model allows to selectively transform SVG graphical data according to the user's profile, thus releasing only the features that the user is entitled to see. In this paper we present the design and implementation of an access control system based on this model. More precisely, in Section 2 we show an overview of SVG. In Section 3, we summarize the access control model for SVG documents presented in [1]. In Section 4, we describe the architecture of the system and the algorithms implementing the enforcement of the access control rules. In Section 5

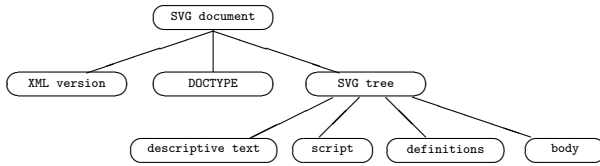


Fig. 1. General Structure of an SVG Document

we illustrate the system’s working through an example. Finally, in Section 6 we present our conclusions and future work.

2 Overview of SVG

An SVG document has a structure, composed of several optional elements placed in the document in an arbitrary order. Figure 1 shows the general structure used. Nodes **XML Version** and **DOCTYPE** are common for any XML-based document and specify the XML version used in the document and information about the type of the document (the public identifier and the system identifier for SVG 1.0), respectively. Node **SVG** contains all the elements specific to SVG documents and is composed of four parts: descriptive text, script, definitions, and body. The descriptive text includes textual information not rendered as part of the graphic and is represented by two elements: `<title>`, usually appearing only once, and `<desc>`, appearing several times to describe the content of each SVG fragment. The script portion contains function definitions. Each function is associated with an action that can be executed on SVG objects in the document. Functions have a global scope across the entire document. The definition portion contains global patterns and templates of graphical elements or graphical properties that can be reused in the body of the SVG document. Each definition is characterized by a name, which is used in the body of the document to reference the definition, and by a set of properties. The graphical elements to be rendered are listed after the `<defs>` node, according to the order of rendering. Each element can belong to any of the basic SVG graphics elements, such as **path**, **text**, **rect**, **circle**, **ellipse**, **line**, **polyline**, **polygon**, and **image**, whose names are self-explanatory. The body of an SVG document contains any number of container and graphics elements. A container element can have graphics elements and other container elements as child elements. Container `<g>` is used for *grouping together* related graphics elements. A graphics element can cause graphics to be drawn. For instance, the **use** graphics element references another element (usually a definition) and indicates that the graphical contents of that element must be drawn at that specific point in the document. Each SVG element may have its own properties, modeled by XML attributes. All elements in the document can be uniquely identified including the special attribute `id=‘identifier’`. It is also possible to include user-defined properties, which can be useful for SVG data processing.

3 An Access Control Model for SVG Documents

The model is based on the use of authorizations (or access control rules) that are themselves expressed with an XML-based language. Each authorization specifies the *subject* to which the authorization applies, the *object* to which the authorization refers, the *action* to which the authorization refers, and the *sign* describing whether the authorization states a permission ($\text{sign} = '+'$) or a denial ($\text{sign} = '-'$). Basically, an authorization $\langle \text{subject}, \text{object}, \text{action}, \text{sign} \rangle$ states that *subject* is allowed ($\text{sign} = '+'$) or is denied ($\text{sign} = '-'$) to perform *action* on *object*.

Subjects. Our model allows authorizations to be referred to specific user identities (e.g., **Sam**), user groups (e.g., **Employee**), and properties of the users (e.g., name, address, and specialty) that are stored in profiles. Each profile is modeled as a semi-structured document and can be referenced by means of XPath expressions [13]. More precisely, the subject field is composed of two parts: (1) the identity of the user/group, and (2) a subject expression, which is an XPath expression on users' profiles. Authorization subjects are then defined as XML elements of the form:

```
<subject>
  <id value='user/group-id' />
  <subj-expr>xpath-expr</subj-expr>
</subject>
```

Objects. Objects are elements of an SVG document. There are three kinds of protection objects: *definitions* (**<defs>**), *groups* (**<g>**), and *SVG elements*. SVG elements can be graphical or textual elements, such as **rect** or **circle**, or can be elements referencing the definitions. The model allows the association of authorizations with any of such specific elements. Objects can be referenced by using both generic XPath expressions on the SVG document and high level predicates that make the reference independent of the syntax of the elements in the document. More precisely, the model defines four ways of identifying objects:

- a *path expression* resolving in the object;
- an *object identifier* (the value of attribute **id**);
- a *type* (the value of attribute **typeElement**);
- a function **perimeter** that can take as input a path expression, an object identifier, or a type and returns the area that contains the object given as input.

Authorization objects are then defined as XML elements of the form:

```
<object>
  <refer value='object-id' />
  <cond>pred-expr</cond>
</object>
```

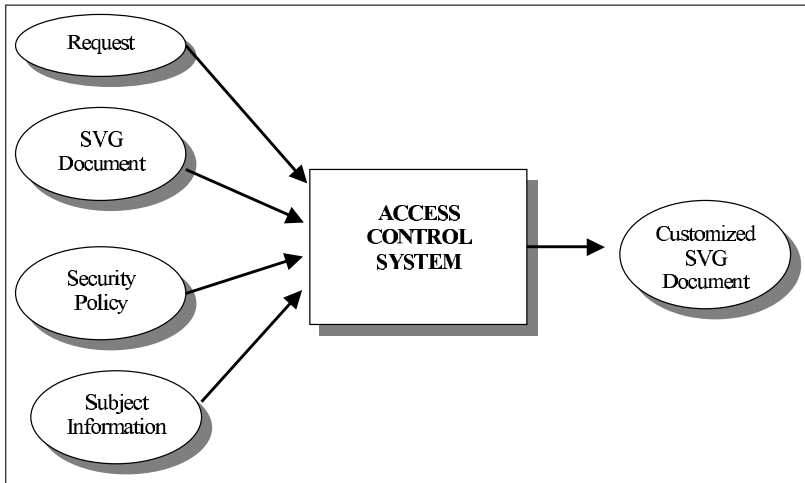


Fig. 2. Hight Level Scheme

This structure allows for specifying an object or a set of objects, and a condition for referencing all elements satisfying specific semantically rich constraints (e.g., computers inside a room). Conditions are boolean expressions that can use the following *predicates*:

- **inside**(*obj*). It returns the object in the authorization if it is inside an element whose identifier, type, or name is *obj*.
- **together_with**(*obj*). It returns the object in the authorization if it is a child of an element together with an object whose identifier, type, or name is *obj*.
- **number_of**(*obj*, *n*). It returns the object in the authorization if there are *n* instances of the object whose identifier, type, or name is *obj*.

Action. Actions can be render, rotate, change color, and so on. For the sake of simplicity, we consider as default action the rendering of an SVG document, so in the authorizations we do not explicitly specify an ‘action’ element.

4 Implementation of the System

The model briefly summarized in this paper has been implemented by using the ASP technology [7] under the Microsoft Internet Information Server [5]. The language used to implement all the algorithms of the system is Visual Basic Script [8]. To manage the XML Document Object Model (DOM) [2], we have selected the parser MSXML [6] that provides a rich set of functions and procedures. Figure 2 illustrates the abstract working of our system. Given an access request, the involved SVG document, the security policy containing all the authorizations defined on the document, and the subject information (i.e.,

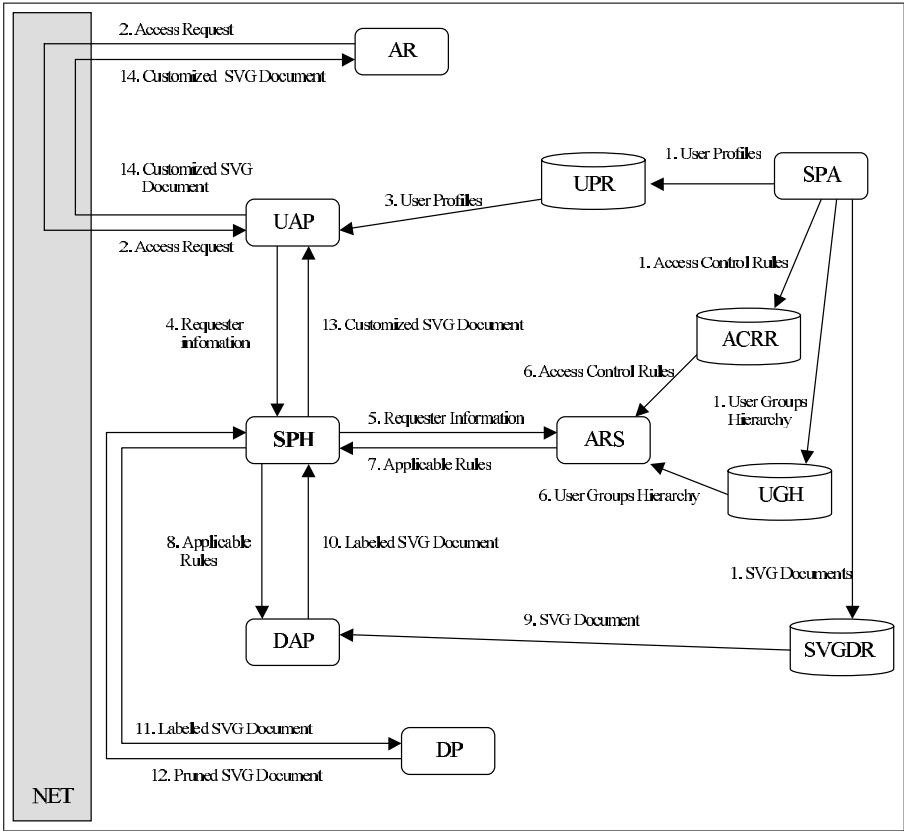


Fig. 3. System Architecture

the user group hierarchy definition and users' profiles), the system has to generate the customized SVG document that includes only the features that the requester is entitled to see.

We now describe the architecture of the system (Section 4.1), and the main algorithms enforcing the security policy (Section 4.2) together with their complexity.

4.1 System Architecture

The architecture of the system (see Fig. 3) is composed of a set of data repositories, a set of modules, and several data flows.

Data repositories. The system includes four basic repositories:

- *User Profiles Repository (UPR)*: It stores the XML-based user profiles.
- *Access Control Rules Repository (ACRR)*: It stores the XML-based documents that contain all the authorizations defined in the security policies.

- *User Groups Hierarchy (UGH)*: It stores the information about the user groups hierarchy.
- *SVG Documents Repository (SVGDR)*: It contains all the SVG documents that have to be protected.

Modules. The modules included in the system are the following.

- *Security Policy Administrator (SPA)*. It creates and manages all the repositories.
 - *Access Requester (AR)*. It manages the access request information.
 - *User Access Point (UAP)*. It is in charge of managing the user authentication.
 - *Security Policy Handler (SPH)*. It coordinates the communication between the modules to fulfill the general goal of the system.
 - *Applicable Rules Selector (ARS)*. It considers all the access control rules specified in the security policy, and selects only those that are applicable to the requester.
- q
- *Document Analyzing Point (DAP)*. It processes the SVG document, propagating permission and denial labels in the SVG tree according to the authorizations selected by the ARS.
 - *Document Pruner (DP)*. It prunes a labeled SVG document, leaving only the SVG features that the requester is entitled to see.

Enforcement. The client-server model enforces the security policy as follows.

1. The SPA creates all repositories (UPR, ACRR, UGH and SVGDR) in the server.
2. The AR sends the access request and the login and password of the requester to the UAP.
3. The UAP reads from the UPR module the requester's profile.
4. The UAP prepares an XML request document that includes the requester's profile and information about the document the requester wants to access. This XML document is sent to the SPH.
5. The SPH coordinates the other modules. It initially sends the requester information to ARS.
6. The ARS reads from the ACRR the access control rules, and from the UGH the user groups hierarchy.
7. Once the ARS has collected the information about the requester, the access control rules, and the user groups hierarchy, it selects the access control rules that are applicable to this particular requester, according to her profile and to the user group membership. The set of applicable access control rules are then sent to the SPH.
8. The SPH sends the applicable access control rules to the DAP.
9. The DAP reads from the SVGDR the SVG document.

10. Once the DAP has the applicable rules and the SVG documents, it starts the labeling process on the SVG document. Each rule is processed, and the corresponding features of the SVG document are labeled. At the end of this process, the labeled document is sent to the SHP.
11. The SPH sends the labeled SVG document to the DP in order to obtain the visible document.
12. The DP prunes all the SVG elements that are denied and sends the resulting document to the SPH.
13. The SPH sends the customized document to the UAP.
14. The UAP offers the rendering of the resulting SVG document.

4.2 System Development

As we have described in the previous section, we can identify four main processes: user information gathering, applicable rules selection, SVG document labeling, and SVG document pruning. In the following, we give some implementation details of these processes.

User Information Gathering Algorithm. The UAP module is in charge of managing the user authentication. To this end, the algorithm needs to access the authentication information in the XML document that contains the user profiles. The corresponding XPath query is:

```
/users/user-profile[login[@value='" + login + "'] and  
password[@value='" + password + "']]
```

Once we have identified the requester, we just have to create a XML document that includes the requester information and the object that she wants to access.

Complexity. Let n be the number of user profiles stored in the system. The complexity of this algorithm is $\Theta(n)$.

Applicable Rules Selection Algorithm. The security policy can include many access control rules and the system has to determine the rules that applies to the requester. To this purpose, the requester information is sent from the SPH to the ARS, which is in charge of reading all the access control rules from the ACRR and the user groups hierarchy from the UGH, and then to filter only those rules that involve the requester. The algorithm is as follows:

1. For each access control rule, we identify all affected users and decides if the requester is one of them. We first consider the **subject id**:
 - If the subject id corresponds to an individual user u and u is the requester, then the rule is considered, otherwise it is discarded.
 - If the subject id corresponds to a group g , and the requester is a direct or indirect member of g , the rule is considered, otherwise it is discarded.

The rules thus selected have to be taken into consideration if and only if the requester's profile satisfies the XPath expression that can be included in such rules.

2. The applicable rules are ordained according to their priority level, from the lowest to the the highest. More precisely, the priority level is established by comparing the **subject id** of the rules: rule **r** has a priority greater than rule **r'** if the **subject id** of **r** is an ascendant of the **subject id** of **r'** in the user group hierarchy. Incomparable rules or with the same level of generality or specificity have the same priority. In case of conflict rules (i.e., a positive and a negative rule both applicable) that are incomparable or with the same level of generality or specificity, the denials take precedence principle is applied. This means that the priority of a negative rule is greater than the priority of a positive rule.

At the end of this process, we obtain an ordered list of applicable rules that has to be sent to the SPH.

Complexity. Let **n** be the number of access control rules and **m** be the number of groups. The complexity of point 1 of the algorithm is $\Theta(n \lg m)$, where $\lg m$ is derived from the binary search of the user groups tree. Point 2 of the algorithm has been implemented by applying the **quick sort** algorithm and therefore the complexity is $\Theta(n \lg n \lg m)$. So, we can conclude that the complexity of the complete algorithm is $\Theta(n \lg n \lg m)$.

SVG Document Labeling Algorithm. The labeling process consists in associating a positive or negative label with each element in the SVG document. Once the SPH sends the ordered list of applicable access control rules to the DAP, the SVG document has to be read from the SVGDR, and all these rules has to be processed in order to decide which elements of the document the requester is entitled to see.

Positive rules (sign='+') generates positive labels associated with the corresponding SVG nodes, and negative rules (sign='-') generates negative labels associated with the corresponding SVG nodes.

The labeling process is basically composed of two steps:

1. *Top-down labeling.* For each rule in the ordered list the following process is applied:
 - Determine the SVG elements that satisfy the object specification.
 - A label with the sign of the rule is associate with each selected SVG element. If some of these elements have already a sign label, it is over-written, because previous labels corresponds to more general rules.
 - Labels are then recursively propagated to all SVG elements that are contained in the affected SVG elements.
2. *Bottom-up labeling.* Whenever a node has a positive label and the corresponding father has a negative label, the labeling process has to go up in the SVG tree, making visible the perimeters of all the containers. This process

is realized by applying a depth-first search: for each SVG node n , if the sign of n is '+' the sign of the perimeter node of all ascendant of n is set to '+'.

The resulting labeled SVG document is then sent to the SPH module to prepare the definitive document.

Complexity. Let n be the number of applicable rules and m be the number of SVG nodes (elements that compose the tree of the SVG document). The complexity of the top-down and bottom-up labeling processes is $\Theta(n*m)$ and $\Theta(m)$, respectively.

SVG Document Pruning Algorithm. The pruning process consists in deleting all nodes of the labeled SVG document that the requester is not entitled to see. In particular, if the security policy is open (i.e., an element is accessible unless a negative rule specifies the opposite), we have to delete all nodes with a negative label. On the other hand, if the security policy is close (i.e., an element cannot be accessed unless a positive rule specifies the opposite), we have to delete all non-labeled nodes and all nodes with a negative label. We consider the open security policy as default. The pruning process is carried out by the DP module, and works as follows:

- The SVG tree is traversed by means of a depth-first search, and for each node:
 - If the node is a simple node (not a container), and its label is '–', we prune the node.
 - If the node is a container and its label is '–', we have to prune the node, but only if all its children have also a negative '–' label. Otherwise, this node remain intact, and the children are processed.

Complexity. Let n be the number of nodes in the SVG tree. The complexity of this algorithm is $\Theta(n)$.

5 An Example

To illustrate the working of the system, consider the simple SVG document in Fig. 4a rendered in Fig. 4b. It represents the map of a Department of Defense (DD). The body of the SVG document is a group element whose identifier is `deptdefense`, and its sub-elements are `main_entrance`, `public_area`, and `private_area`. Group `public_area` includes `cafeteria`, `IDoffice`, two `restroom` instances, `info_desk`, and `public_affair` office. Group `private_area` includes `emerg_unit`, `navy` and `air control`, `computer_room`, four `videocamera` instances, eight `laser_sensor` instances, and two `alarm_control` instances. Each of these elements are composed of a graphical representation and a `name`. Consider also the user groups hierarchy in Fig. 5a. Here, the root element `users` has two subgroups: `Non DD Members` and `DD`

```

<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" .....
<svg width="600" height="600">
  <defs>
    <rect id="rectRoom" width="150" height="132" style="stroke:rgb(0,0,0);fill:rgb(255,255,0)"/>
    <symbol id="computer" viewBox="0 0 25 25">.....</symbol>
  </defs>
  <g id="deptDefense">
    <g id="perimeterdeptDefense" perimeter="yes" typeElement="Perimeter">
      <polygon ...../>
    </g>
    <g id="PublicArea">
      <g id="Cafeteria">
        <g id="perimeterCafeteria" perimeter="yes" typeElement="Perimeter">
          <use x="450" y="0" xlink:href="#rectRoom"/>
        </g>
        <text x="470" y="70" style="font-size:20"> Cafeteria </text>
      </g>
    </g>
    <g id="PrivateArea">
      <g id="EmergencyUnit">
        <g id="perimeterEmergencyUnit" perimeter="yes" typeElement="Perimeter">
          <use x="0" y="0" xlink:href="#rectRoom"/>
        </g>
        <text x="5" y="70" style="font-size:20"> Emergency Unit </text>
        <g id="LaserSensorsEU">
          <use x="10" y="10" xlink:href="#LaserSensor" typeElement="security"/>
        </g>
      </g>
      <g id="NACControl">
        <g id="perimeterNACControl" perimeter="yes" typeElement="Perimeter">
          <use x="0" y="132" xlink:href="#rectRoom"/>
        </g>
        <text x="5" y="180" style="font-size:20"> Navy and Air </text>
        <text x="20" y="200" style="font-size:20"> Control </text>
        <g id="radarControl">
          <rect id="radarControlrect" ...../>
          <text x="15" y="230" style="font-size:15"> Radar Control </text>
        </g>
        <g id="LaserSensorsNAC">
          <use x="10" y="140" xlink:href="#LaserSensor" typeElement="security"/>
        </g>
      </g>
    </g>
    <g id="ComputerRoom">
      <g id="perimeterComputerRoom" perimeter="yes" typeElement="Perimeter">
        <use x="0" y="265" xlink:href="#rectRoom"/>
      </g>
      <text x="0" y="330" style="font-size:20">Computer Room </text>
      <g id="computers">
        <use x="20" y="270" width="30" height="30" xlink:href="#computer" typeElement="computer"/>
      </g>
      <g id="LaserSensorsCR">
        <use x="10" y="270" xlink:href="#LaserSensor" typeElement="security"/>
      </g>
    </g>
  </svg>

```

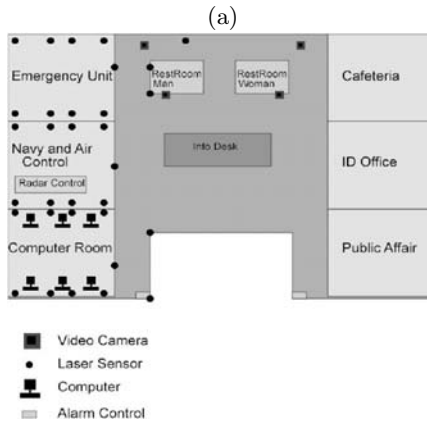


Fig. 4. An example of SVG document (a) and its rendering (b)

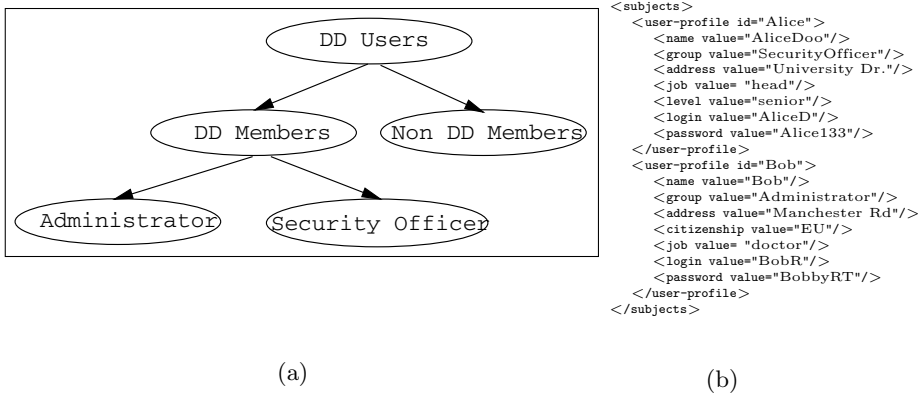


Fig. 5. An example of user group hierarchy (a) and two examples of user profiles (b)

Members that in turn has two subgroups, namely **Administrator** and **Security Officer**. Figure 5b shows the profile of two users, **Alice** and **Bob**. These profiles show among other details, that **Alice** is a member of the **Security Officer** group and **Bob** is a member of the **Administrator** group.

Suppose now that the access to the given SVG document has to be regulated according to the following three authorizations.

1. Everybody can see the content of any room in the public area.

```

<authorization-rule>
  <subject>
    <id value="users"/>
  </subject>
  <object>
    <refer value = "id.PublicArea"/>
  </object>
  <sign>+</sign>
</authorization-rule>

```

2. Members of the **Security Officer** group whose job is not **controller** cannot see computers.

```

<authorization-rule>
  <subject>
    <id value="SecurityOfficer"/>
    <subj-expression>
      /subjects/user-profile[job[not(@value='controller')]]
    </subj-expression>
  </subject>
  <object>
    <refer value = "typeElement.computer"/>
  </object>
  <sign>-</sign>
</authorization-rule>

```

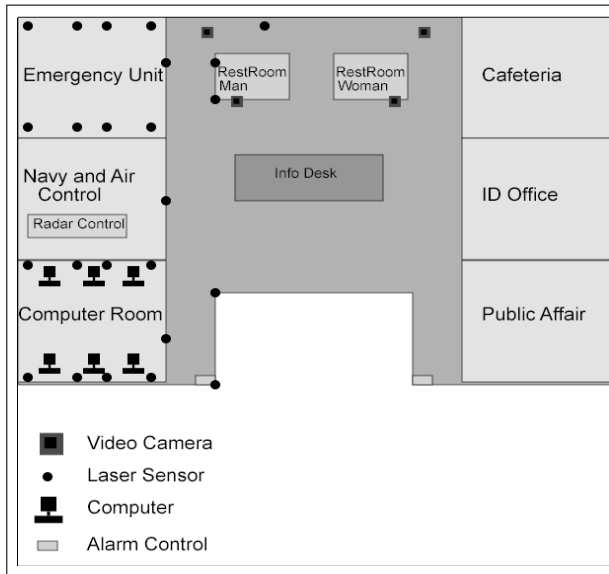


Fig. 6. Final SVG rendering

- Members of the **Administrator** group cannot see the security elements in the **navy and air control** room.

```
<authorization-rule>
  <subject>
    <id value="Administrator"/>
  </subject>
  <object>
    <refer value = "typeElement.security"/>
    <cond>inside(id.NAControl)</cond>
  </object>
  <sign>-</sign>
</authorization-rule>
```

Consider now a request to read the map of the department submitted by user Bob. The **applicable rules selection algorithm** selects the first rule and the third rule; the second rule does not match the profile of Bob. Once the applicable rules have been selected, the **labeling algorithm** includes the authorization labels in the SVG document, and finally, the **pruning algorithm** eliminates all the SVG documents that Bob is not entitled to see. Figure 6 illustrates the portion of the map returned to Bob. As you can notice, according to the first rule, Bob is entitled to see the public area, and, according to the third rule, Bob cannot see all the security elements included in the **navy and air control** room.

6 Conclusions and Future Work

We have presented the design and implementation of an access control system for SVG documents. We have analyzed the architecture, and the algorithms used in the system. The system allows for controlling access to any SVG document in a user-transparent way and is currently used (in a experimental way) for controlling access to a graphical representation of the building of a computer science faculty.

Future work includes the implementation of an administrative tool for managing all the system components (access control rules, users, groups, and so on). Another important aspect is the extension of the model to control access to other XML-based multimedia standards formats, such as SMIL [11] for multimedia presentations, VoiceXML [12] for dialog, and MPEG-21 [3] and MPEG-7 [4] for video.

References

1. E. Damiani, S. De Capitani di Vimercati, E. Fernandez-Medina, and P. Samarati. An access control system for svg documents. In *Proc. of the Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, University of Cambridge, UK, July 2002.
2. Philippe Le Hégaret. *DOM*. W3 Consortium, June 2002. <http://www.w3.org/DOM/>.
3. International Organisation for Standardisation. *MPEG-21 Overview v.5*, October 2002. <http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm>.
4. International Organisation for Standardisation. *MPEG-7 Overview v.8*, July 2002. <http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm>.
5. Microsoft Corporation. *Internet Information Server*. www.microsoft.com/iis.
6. Microsoft Corporation. *MSXML 3.0 SDK*. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk30/hm/xmmscxmloverview.asp>.
7. Microsoft Corporation. *Active Server Pages*, December 2000. <http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522>.
8. Microsoft Corporation. *VB Script fundamentals*, 2003. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbstutor.asp>.
9. P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.
10. World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001.
11. World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL 2.0)*, August 2001. <http://www.w3.org/TR/smil20>.
12. World Wide Web Consortium. *Voice Extensible Markup Language (VoiceXML) Version 2.0*, April 2002. <http://www.w3.org/TR/voicexml20>.
13. World Wide Web Consortium (W3C). *XML Path Language (XPath) 2.0*, December 2001. <http://www.w3.org/TR/xpath20>.

Administration Model for Or-BAC

Frédéric Cuppens¹ and Alexandre Miège²

¹ GET/ENST Bretagne/Département RSM,
BP 78, 2 rue de la Châtaigneraie,
35512 Cesson Sévigné Cedex, France
`frederic.cuppens@enst-bretagne.fr`

² ENST,
46, rue Barrault, 75634 Paris Cedex 13, France
`alexandre.miege@cert.fr`

Abstract. Even though the final objective of an access control model is to provide a framework to decide if actions performed by subjects on objects are permitted or not, it is not convenient to directly specify an access control policy using concepts of subjects, objects and actions. This is why the Role Based Access Control (RBAC) model suggests using a more abstract concept than subject to specify a policy. The Organization Based Access Control (Or-BAC) model further generalizes the RBAC model by introducing the concepts of activity and view as abstractions of action and object. In the Or-BAC model, it is also possible to specify privileges that only apply in some given contexts.

In this paper, we present AdOr-BAC, an administration model for Or-BAC. This model is fully homogeneous with the remainder of Or-BAC. AdOr-BAC can control assignment of user to role (User Role Administration), assignment of permission to role (Permission Role Administration) and assignment of user to permission (User Permission Administration). This last possibility is useful to control fine grained delegation, when a user wants to grant a specific permission to another given user. AdOr-BAC is compared with other administration models, such as the ARBAC model suggested for RBAC, showing some of its advantages.

1 Introduction

The final objective of an access control policy is to specify the *permissions*, *obligations* and *prohibitions* that control the *actions* performed by *subjects* on *objects*. However, when defining its access control policy, an *organization* does not directly specify that a given subject (for instance John) is permitted to perform a given action (for instance *read*) on a given object (for instance Jack's medical record). Organization based access control policies never mention that they apply to specific subjects, actions or objects. Instead, they use more abstract concepts such as the concept of role [14,7]. In this case, the access control policy does not directly grant permissions to subjects but to roles. A given subject will then obtain permissions by playing roles, in which case this user will inherit all the permissions associated with these roles. In [3], the Organization Based Access

Control (Or-BAC) model is defined and it is argued that similar abstractions should be associated with actions and objects. For this purpose, the Or-BAC model introduces the abstract concepts of activities and views. In Or-BAC, the access control policy defines permissions (or obligations or prohibitions) that control the *activities* performed by *roles* on *views*. For instance, the policy might specify that role *physician* has permission to perform activity *consult* on view *medical record*¹. We can then derive that user John is permitted to perform action read on object Jack_med_record if John is playing the role physician, read is an action that corresponds to the activity consult and object Jack_med_record belongs to the view medical record.

A complete access control model must provide an administration model. For instance, the Role Based Access Control (RBAC) model is associated with the ARBAC97 model [11], further refined in the ARBAC99 [13] and ARBAC02 [9] models. The ARBAC model includes two main components: (1) The URA (User Role Administration) model to control who is permitted to assign a user with a new role and who is permitted to revoke a user from an existing role, (2) The PRA (Permission Role Administration) model to control who is permitted to assign a role with a new permission and who is permitted to revoke a role from an existing permission. The RBAC model also includes the RRA component for Role-Role Administration to manage the role hierarchy. The objective of this paper is to present an administration model for the Or-BAC model called AdOr-BAC. The AdOr-BAC model includes three main components: URA, PRA and UPA (for User Permission Administration). The objective of the URA and PRA components is similar to components with similar names already defined in the RBAC model, but the model we suggest in AdOr-BAC is different. This model is fully homogeneous with the remainder of Or-BAC. In particular, the syntax used to specify permissions in the URA and PRA models is similar to the one suggested in Or-BAC. We shall see that this approach has several advantages over the ARBAC model. The UPA component does not exist in the ARBAC model. It is useful when a given user wants to grant a permission to another given user. For instance, John (a physician) may want to grant to Jane (his secretary) a permission to have an access to Jack's medical record. The UPA model applies in this case. It is used to specify that subjects playing the role of physician are permitted to grant to other subjects playing the role of medical secretary a permission to have an access to objects belonging to the view medical record. Notice that, using UPA, we can also specify that subjects playing the role physician are forbidden to grant a permission to subjects playing another role than medical secretary, for instance journalist or insurer. Notice also that the PRA model does not apply in this case since the general permissions of role medical secretary do not change, only Jane's permissions are updated.

The remainder of this paper is organized as following. In section 2, we present ARBAC, the administration model suggested for RBAC, and discuss some of its

¹ Actually, the Or-BAC model allows the administrators to specify more complex permissions since one can consider that each permission only applies in some given *contexts* (see section 3 for further details).

weaknesses. Section 3 briefly recalls the Or-BAC model. Section 4 presents the AdOr-BAC model that is used to administer the Or-BAC model. Finally section 5 concludes the paper.

2 ARBAC

The Role-Based Access Control (RBAC) model [6,8,14] aims to use the *role* as a central concept. Ravi Sandhu proposed an administration model, ARBAC, dedicated to the management of a RBAC policy.

ARBAC97: ARBAC97 [11] is the first RBAC administration model. ARBAC has two main features. First, it provides the possibility of administrating an RBAC policy in a decentralized, but without losing the control over rights' propagation. Second, though the administrative roles and permissions are based on RBAC, they are completely separated from the regular roles and permissions. ARBAC97 provides three sub-models:

- URA97 [10]. This model describes how to assign users to the predefined roles. The assignment by an administrative role of a user to a regular role is based on a ternary relation “can_assign” between the administrative role, the prerequisites roles and the regular role. That is, a member of an administration role can assign a user to a regular role if this user satisfies the condition corresponding to the prerequisite roles.
- PRA97 [11]. This model is the dual of URA97 and it describes the assignment of permissions to roles. It is also based on a ternary relation with prerequisite conditions.
- RRA97 [12]. This last model proposed rules for the role-role assignment, that is the construction of the role hierarchy.

Therefore, ARBAC97 offers a proper administration model which is not exactly the case for the other security models. In order to obtain a decentralized administration of a RBAC policy, it could be used this way: The management of the role hierarchy and the assignment of the permissions is carried out by a centralized authority on the one hand. On the other hand, the assignment and the revocation can be left under the responsibility of the chiefs of the different departments or units, through the assignment of these chiefs to administrative roles.

However, it is noteworthy to point out the following shortcomings. ARBAC claims that it is an auto-administrated model. This is not completely true because it does not use the RBAC model to define administrative permissions. Instead, it creates new assignment and revocation rules (such as `can_assign` and `can_revoke`) used by the administrative roles. These rules are distinct from the approach suggested in RBAC to define permissions associated with regular roles. As we mentioned, the assignment relation is ternary. Thus the prerequisite conditions depend on the administrative role and the regular role. However, it seems

that the prerequisite conditions generally depend logically and only on the regular role and should rather be considered as a constraint on the regular role.

Moreover, ARBAC does not give any information on the creation of the roles, and does not offer any delegation mechanism. Since ARBAC97, a proposal has been made to manage delegation in RBAC model through RBDM [1]. Delegation will be further discussed in section 4.4. ARBAC does not offer means to express contextual conditions. Thus, it is not possible to express that a given administrative role is permitted to assign a permission to a regular role only at working hours or only from his own terminal. This kind of restriction can be usefull to detect administrator's abuse of power for instance.

ARBAC Extensions: Two ARBAC extensions have been proposed. ARBAC99 [13] presents a way to manage the mobile and immobile users and permissions. Unlike a mobile user, an immobile user can be seen as a non-permanent user such as a user under training, a visitor, a consultant, etc. In this case, the user can be a member of a role and get the corresponding permissions. But an administrative role cannot use this membership to put the immobile user into other roles. That is, an immobile user cannot climb the hierarchy. The same idea is used for the immobile permissions.

The objective of ARBAC02 [9] is different. Several weaknesses of ARBAC97 have been pointed out. Through ARBAC02 some improvements were proposed to resolve, among others, the multi-step user assignment which generate a lot of work for the security officers and which causes redundant tuples in the URA management. The main modification made in ARBAC02 affects the prerequisite conditions for the user and the permission assignment. An organization structure of user pool and an organization structure of a permission pool are created. The first one is managed by the human resources group, the second one by the IT group. We obtain two hierarchies independent from the role hierarchy. User and permission assignment is made by the security officers by picking user and permissions in these pools. This simplifies the assignment processes.

These two extensions of ARBAC97 are interesting but do not answer the shortcomings we have just mentioned. Moreover ARBAC02 simplifies the assignment process, but transfer the problem of the prerequisite conditions onto the human resources group and the IT group.

SARBAC (Scoped Administration of Role-Based Access Control) [2] suggests an extension of RRA97, called RHA₄, and an alternative to ARBAC97. SARBAC relies on administrative scope which changes dynamically as the role hierarchy changes. Thus, update operations over RBAC96 and SARBAC relations become easier and cannot lead to inconsistent rules. In particular, SARBAC makes it possible to delete a role without any restriction. Unlike in ARBAC97, it is possible to assign administrative roles to users as SARBAC does not make any distinction between regular and administrative roles.

3 Or-BAC

Before presenting the administration model for the Or-BAC model, we shall briefly recall the main components of this model (see [3] for further details). The most important entity in Or-BAC is the entity *Organization*. Roughly speaking, an organization can be seen as an organized group of subjects, playing some role or other. Notice that a group of subjects does not necessarily correspond to an organization. More precisely, the fact that each subject plays a role in the organization corresponds to some agreement between the subjects to form an organization.

In the organization, subjects will request to perform actions on objects and, as mentioned in the introduction, the final objective of an access control policy is to decide if these requests are permitted or not. In the Or-BAC model, a subject will be either an active entity, i.e. a user, or an organization. Actions will mainly correspond to concrete computer actions such as “read”, “write”, “send”, etc.

However, permission in the Or-BAC model does not directly apply to subject, action and object. Instead, subject, action and object are respectively abstracted into role, activity and view. A view corresponds to a set of objects that satisfy a common property. Similarly, an activity will join actions that partake of the same principles.

A given access control policy is then specified by a set of *facts* having the form:² *Permission(org, role, activity, view, context)*. These facts specify that, in organization *org*, a given *role* is permitted to perform a given *activity* on a given *view* in a given *context*. Examples of context may be *Night*, *Working-Hours* or *Urgency* (see section below for further details about the context definition).

Specifying the access control policy by facts is an important difference compared with other approaches based on logical *rules*, such as Ponder [5]. This will represent a major advantage when we shall define how to administer Or-BAC (see section 4). Notice that the specification of the security policy is parameterized by the organization so that it is possible to handle simultaneously several security policies associated with different organizations.

Basic Concepts of Or-BAC: In Or-BAC, there are eight basic sets of entities: *Org* (a set of organizations), *S* (a set of subjects), *A* (a set of actions), *O* (a set of objects), *R* (a set of roles), *A* (a set of activities), *V* (a set of views) and *C* (a set of contexts).

We shall assume that $Org \subseteq S$ (that is any organization is a subject) and that $S \subseteq O$ (that is any subject is an object). Any entities in the Or-BAC model may have some attributes. This is represented by functions that associate the entities with the value of these attributes. For instance, if *s* is a subject, then *name(s)* represents the name of *s*, *address(s)* its address, etc.

² Actually, in [3], it is also possible to specify prohibitions and obligations using Or-BAC. Here, for the sake of simplicity, we shall only consider permissions. This is mainly to eliminate the problem of conflicts between permission and prohibition. However, we plan to analyze this problem of conflict in a forthcoming paper.

Modelling the Organization Components: In the organization, subjects are empowered in roles, objects are used into views and actions fall within activities. This is represented by the following relationships:

- *Empower* is a relation over domains $Org \times S \times \mathcal{R}$. If *org* is an organization, *s* a subject and *r* a role, then $Empower(org, s, r)$ means that *org* empowers subject *s* in role *r*. Unlike the TMAC model or the RBAC model which consider binary relations between organizations and subjects or between subjects and roles, notice that our model consider a ternary relation between organizations, subjects and roles. This is useful to model situations where a given subject plays several roles but in different organizations. Let us also remark that subjects might be users as well as organizations.
- *Use* is a relation over domains $Org \times O \times \mathcal{V}$. If *org* is an organization, *o* is an object and *v* is a view, then $Use(org, o, v)$ means that *org* uses object *o* in view *v*. This ternary relation makes ourselves able to characterize organizations that give different definitions to the same view. For instance, take the case of the view “medical record” defined in Purpan hospital as a set of Word documents and defined in Rangueil hospital as a set of tuples in a relational database.
- *Consider* is a relation over domains $Org \times A \times \mathcal{A}$. If *org* is an organization, α is an action and *a* is an activity, then $Consider(org, \alpha, a)$ means that *org* considers that action α falls within the activity *a*. Since *Consider* is a ternary relation, different organizations may decide that one and the same action comes under distinct activities or that different actions come under the same activity. For instance, activity “consulting” corresponds, in Purpan hospital, to an action “read” that can be ran on data files whereas it corresponds, in Rangueil hospital, to action “select” that can be performed on relational databases.

Context Definition: Contexts are used to specify the concrete circumstances where organizations grant roles permissions to perform activities on views. In the health care domain, the entity *Context* will cover circumstances such as “urgency”, “industrial medicine”, “attending physician”, etc. Every context can be seen as a ternary relation between subjects, objects and actions defined within a given organization. Therefore, entities *Organization*, *Subject*, *Object*, *Action* and *Context* are linked together by the relationship *Define*:

- *Define* is a relation over domains $Org \times S \times A \times O \times \mathcal{C}$. If *org* is an organization, *s* is a subject, α is an action, *o* is an object and *c* a context, then $Define(org, s, \alpha, o, c)$ means that within organization *org*, context *c* holds between subject *s*, action α and object *o*.

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions will be formally specified by logical rules. For instance, we may define the context *Night* as follows:³

³ In the remainder of this paper, we shall use a logical notation to represent relationship: if *R* is a *n*-ary relationship over domains $D_1 \times \dots \times D_n$, then the predicate $R(d_1, \dots, d_n)$ is *true* if and only if $\langle d_1, \dots, d_n \rangle \in R$.

- $\forall s, \forall \alpha, \forall o, \forall c_1, \forall c_2, (Define(H1, s, \alpha, o, Night) \leftrightarrow (20 : 00 \leq time(global\_clock) \vee time(global\_clock) \leq 8 : 00))$

that is, in H1, the context “night” is *true* between subject s , action α and object o between 20:00 and 8:00. In the following, we shall use another context called “default”. This context is *true* in every circumstance. It is defined as follows:

- $\forall org, \forall s, \forall \alpha, \forall o, Define(org, s, \alpha, o, Default)$

that is, in every organization org , the context “default” is always *true* between subject s , action α and object o .

Policy Definition: In the Or-BAC model, the access control policy is defined using the relationship *Permission* as follows:

- *Permission* is a relation over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$. If org is an organization, r is a role, a is an activity, v is a view and c a context then $Permission(org, r, a, v, c)$ means that organization org grants role r permission to perform activity a on view v within context c .

Deriving Concrete Permission: The relationship *Permission* enables a given organization to specify permissions between roles, activities and views in a given context. However, an access control model must provide a framework for describing the concrete actions that may be performed by subjects on objects. For the purpose of modelling concrete permissions, we introduce the relationship *Is_permitted* as a relationship between subjects, actions and objects:

- *Is_permitted* is a relation over domains $S \times A \times O$. If s is a subject, α is an action and o is an object then $Is\_permitted(s, \alpha, o)$ means that subject s is permitted to perform action α on object o .

In our model, triples that are instances of the relationship *Is_permitted* are logically derived from permissions granted to roles, views and activities by the relationship *Permission*. This is modelled by the following general rule:

- $\forall org, \forall s, \forall o, \forall \alpha, \forall r, \forall v, \forall a, \forall c,$
 $Permission(org, r, a, v, c) \wedge$
 $Empower(org, s, r) \wedge Use(org, o, v) \wedge Consider(org, \alpha, a) \wedge$
 $Define(org, s, \alpha, o, c) \rightarrow Is\_permitted(s, \alpha, o)$

that is, if organization org , within the context c , grants role r permission to perform activity a on view v , if org empowers subject s in role r , if org uses object o in view v , if org considers that action α falls within the activity a and if, within org , the context c is *true* between s , α and o then s is permitted to perform α on o .

Notice that we do not assume that all instances of relationship *Is_permitted* comes from the specification of relationship *Permission*. This means that there may exist other instances of relationship *Is_permitted*. These instances may be viewed as exceptions to the general security policy specified by the relationship *Permission*. This will be used in UPA (see section 4.4) when a user wants to grant a specific permission to another given user.

4 AdOr-BAC: An Administration Model for Or-BAC

4.1 Introduction

The objective of this section is to define an administration model for Or-BAC, called AdOr-BAC. A complete administration model should provide means to control the following activities: management⁴ of organizations, management of roles, activities, views and contexts, assignment (and revocation) of users to roles, assignment (and revocation) of permissions to roles, assignment (and revocation) of users to permissions.

Due to space limitation, we focus in this paper on the user-role assignment, the permission-role assignment and the user-permission assignment. The approach we suggest in AdOr-BAC is to define these administration functions by considering three different views respectively called URA, PRA and UPA. Each organization will manage such views. Objects belonging to these views have specific semantics; namely they will be respectively interpreted as an assignment of user to a role, a permission to role and a permission to a user.

Intuitively, inserting an object in these views will enable an authorized user to respectively assign a user to a role, assign a permission to a role or assign a permission to a user. Conversely, deleting an object from these views will enable a user to perform a revocation.

Defining the administration functions in AdOr-BAC then corresponds to define which roles is permitted to have an access to views URA, PRA and UPA, or to more specific views when the role has not a complete access to one of these views. For instance, the role physician may be only permitted to assign users to the role medical secretary. In this case, the role physician will have not a complete access to the view URA, but only to the sub-part corresponding to the role medical secretary.

The approach we suggest is homogeneous with the remainder of the Or-BAC model. The syntax we use in AdOr-BAC to define permission to administer the policy is completely similar to the remainder of Or-BAC. Actually, strictly speaking, it is even incorrect to consider that AdOr-BAC is a distinct model from Or-BAC. Since, we have simply to consider three new views, namely URA, PRA and UPA in the Or-BAC model, it would be more appropriate to say that Or-BAC is an auto-administered model. In the following we shall present the structure of these three views and further analyze the administration functions associated with management of these views.

Notice that, in the ARBAC model, there are two types of fully separated roles called regular roles and administration roles. In some circumstances this separation is superfluous. For instance the role physician may hold a plurality of administrative and non administrative permissions. In such case, it is not necessary to create two roles, this is, a role *physician* and a role *administration_physician*. The AdOr-BAC model does not impose to create these two roles. But, as a security policy designer could legitimately want to separate them anyway, because

⁴ By manage, we mean create, delete and update.

of separation of duty and least privilege questions, the AdOr-BAC model makes it possible to do so. Thus, we leave such separation optional in the AdOr-BAC model. Keeping this separation makes AdOr-BAC compliant with ARBAC.

4.2 URA in AdOr-BAC

The View *URA*: The aim of the user-role administration is to determine who is allowed to assign a user to a role and on which conditions. Assigning a user to a role equals adding a new object in a given view called *URA*. Three attributes are associated with this view: *subject* to designate the subject which is related to the assignment, *role* that corresponds to the role to which the subject will be assigned and *org* to represent the organization in which the subject is assigned.

For example, if a security officer is allowed to assign a user to the role *physician* in the cardiology department *cardio_dpt* of its hospital *H*, we create the *URA_physician_cardio_dpt* view defined as follows:

$$\begin{aligned} & - \forall ura, Use(H, ura, URA\_physician\_cardio\_dpt) \rightarrow \\ & \quad Use(H, ura, URA) \wedge role(ura) = physician \wedge org(ura) = cardio\_dpt \end{aligned}$$

There is a link between the object belonging to the view *URA* and the relationship *Empower*. This link is modelled through the following rule:

$$\begin{aligned} & - \forall org, \forall ura, Use(org, ura, URA) \\ & \quad \rightarrow Empower(org(ura), subject(ura), role(ura)) \end{aligned}$$

It means that a user empowered in a given organization corresponding to *org* can manage user-role assignment of another organization (corresponding to *org(ura)*). For instance, *org* might be the human resources department of a given company and *org(ura)* might be the different departments of this company.

The Activity *manage*: The view *URA* makes it possible to model the assignment of a user to a role. We have to consider now the activity that corresponds to the permission of assigning someone. We call *assign* this activity. The permission granted to the role *sec_officer* to assign a user to the role *physician* in the department of cardiology *cardio_dpt* of its hospital *H* is expressed as follows: *Permission(H, sec_officer, assign, URA_physician_cardio_dpt, Default)*.

Up to now, we have only dealt with assignment but not with revocation. Notwithstanding it is easy to create the activity *revoke* in the way as the activity *assign*: *Permission(H, sec_officer, revoke, URA_physician_cardio_dpt, Default)*.

When a role is authorized to both assign and revoke users to a specific role, we create the activity *manage*, and consider the activities *assign* and *revoke* as two sub-activities of *manage*:

$$\begin{aligned} & - \forall org, \forall role, \forall view, \forall context, \\ & \quad Permission(org, role, manage, view, context) \rightarrow \\ & \quad \quad Permission(org, role, assign, view, context) \wedge \\ & \quad \quad Permission(org, role, revoke, view, context) \end{aligned}$$

that is if a given role is permitted to manage a given view in a given context, this role is also permitted to perform assignment and revocation of this view in the same context.

The Prerequisite Conditions: In the ARBAC model, the relation *can_assign* makes it possible to add prerequisite conditions on the role of the user concerned by the assignment. It is possible to express this kind of condition in the AdOr-BAC model. Let us consider the following example:

- The director is permitted to designate a user as the head of department of cardiology but only if this user is a member of the role *physician*:
 $Permission(H, director, assign, URA\_head\_cardio\_dpt, Default)$
 The view *URA_head_cardio_dpt* is defined as follows:
 $\forall ura, Use(H, ura, URA\_head\_cardio\_dpt) \leftrightarrow$
 $Use(H, ura, URA\_head\_dpt) \wedge$
 $Empower(H, subject(ura), physician)$

We can thus specify that for the department of cardiology the head must be a physician. It is no use having any prerequisite condition for the revocation of this head, that is why this last permission is granted just for the activity *assign*. This is specified in a similar way as permission *P1* but the permission only applies to activity *assign*.

The user-role assignment in AdOr-BAC is very flexible. A large number of conditions can be expressed such as the prerequisite conditions of ARBAC, thanks to the use of views which make it possible to model the assignments.

4.3 PRA in AdOr-BAC

In the previous section we dealt with the user-role administration. We discuss here the permission-role administration. As we have just seen, we modelled user assignment with the view *URA*. Here, the permission assignment is modelled with a new view called *PRA*. Giving a new permission to a role corresponds to create a new object which complies with the view *PRA*.

The View *PRA*: Five attributes are associated with the view *PRA*:

- *issuer*: the organization where the permission applies
- *grantee*, *privilege*, *target*: role, activity and view concerned by the permission
- *context*: designate the context in which the rule can be applied

There is a link between the objects belonging to the view *PRA* and the relationship *Permission*. This link is modelled as follows :

- $\forall org, \forall context, Use(org, pra, PRA) \rightarrow$
 $Permission(issuer(pra), grantee(pra),$
 $privilege(pra), target(pra), context(pra))$

The Activity *manage*: The same activities *assign*, *revoke* and *manage* defined in the previous section are used to express the authorization given to a role to assign and revoke permissions to other roles.

The Prerequisite Conditions: The prerequisite conditions defined in AR-BAC related to the permission-role assignment can be expressed in our model through the view *PRA* as we saw in the URA section.

4.4 UPA in AdOr-BAC

The URA and PRA components respectively allow an authorized user to assign users to roles and permissions to roles. Thus, these components indirectly enable this authorized user to assign permissions to users. We argue that sometimes a more direct process should enable a user to grant a permission to another user. For instance, let us consider a situation where there are two users, John a physician and Jane his medical secretary. The role medical secretary is not permitted to have an access to the view medical record. John makes a consultation on Jack, a patient and, after this consultation, wants to update Jack's medical record. However, John is too busy to do so; he decides to grant Jane a permission to update Jack's medical record. Notice that permissions of the role medical secretary do not change, Jane simply gets a new permission from John. This is the objective of the UPA component to control the assignment of a new permission to a user and revocation of an existing permission. For this purpose, we consider the same activities *assign*, *revoke* and *manage* as the ones suggested in URA and PRA. Actually, we can consider two different cases called UPA1 and UPA2. UPA1 enables an authorized user to grant another user a permission to perform a specific action on a specific object. UPA2 is more general. It enables an authorized user to grant another user a permission to perform a given activity on a given view. Due to space limitation, we only present UPA1; UPA2 can be similarly defined. We shall then analyze how UPA1 applies to model the concept of delegation.

UPA1: Granting Permissions on Specific Objects and Actions: In this case, we consider a view *UPA1* with five attributes having the same names as PRA but with slightly different meaning: *issuer* represents the organization who is issuing the permission, *grantee* is the subject who is receiving the permission, *privilege* represents the action the grantee is authorized to perform, *target* represents the object the grantee is authorized to have an access to and *context* is the context in which the permission applies. There is a rule that specifies that we can derive, from objects belonging to the view *UPA1*, the fact that a subject is permitted to perform an action on an object. This is modelled by the following rule:

$$- \forall org, \forall upa, \\ Use(org, upa, UPA1) \wedge$$

$$\begin{aligned}
& \text{Define}(\text{issuer}(upa), \text{grantee}(upa), \\
& \quad \text{privilege}(upa), \text{target}(upa), \text{context}(upa)) \\
& \rightarrow \text{Is\_permitted}(\text{grantee}(upa), \text{privilege}(upa), \text{target}(upa))
\end{aligned}$$

that is, if an object *upa* is used by a given organization *org* in view *UPA1* and the issuer of *upa* defines that the context holds between the grantee, the privilege and the target specified by *upa*, then the grantee is permitted to use his privilege on the target. The permissions derived from this rule may be viewed as exceptions to the general permissions defined by the predicate *Permission*. This is exactly the purpose of the UPA component to provide means to specify such exceptions.

Let us now show how this material is used to specify that, in a given hospital H1, a physician is permitted to grant his or her medical secretary a permission to update the medical record of one of his or her patient. We have first to consider a sub-view *SPUMR* (for Secretary Permission of Update Medical Record) of view *UPA1* defined as follows:

$$\begin{aligned}
- \forall upa, \text{Use}(H1, upa, SPUMR) \leftrightarrow \\
\quad \text{Use}(H1, upa, UPA1) \wedge \\
\quad \text{Empower}(H1, \text{grantee}(upa), \text{medical\_secretary}) \wedge \\
\quad \text{Consider}(H1, \text{privilege}(upa), \text{update}) \wedge \\
\quad \text{Use}(H1, \text{target}(upa), \text{medical\_record})
\end{aligned}$$

that is object *upa* is used in view *SPUMR* if and only if it is used in view *UPA1* and the values of attributes *grantee*, *privilege* and *target* respectively correspond to a user empowered as a medical secretary, an action considered as an updating activity and an object used as a medical record. The permission is then specified as follows:

$$- \text{Permission}(H1, \text{physician}, \text{assign}, SPUMR, PSP)$$

that is, in H1, the role *physician* has permission to assign a permission belonging to view *SPUMR* in context *PSP* (for Physician's Secretary and Patient). The context *PSP* is defined as follows:

$$\begin{aligned}
\forall s, \forall a, \forall upa, \text{Define}(H1, s, a, upa, PSP) \leftrightarrow \\
\quad \text{Empower}(H1, s, \text{physician}) \wedge \text{Use}(H1, upa, UPA1) \wedge \\
\quad \text{grantee}(upa) \in \text{secretary}(s) \wedge \text{name}(\text{target}(upa)) \in \text{patient}(s)
\end{aligned}$$

that is, H1 defines that subject *s* performs action *a* on object *upa* in context *PSP* if *s* is a physician in H1, *upa* is used in view *UPA1*, the grantee of *upa* is a secretary of *s* and the target name of *upa* is a patient of *s*.

Using this permission, John (a physician of H1), is permitted to grant Jane (his secretary) a permission to update Jack's medical record (his patient).

Application to Delegation: Modelling delegation is a complex problem. The analysis performed in [1] shows that there are several subtleties leading to many possible definitions of the concept of delegation. The objective of this paper is not to fully investigate this problem. We shall simply show that the expressiveness

of AdOr-BAC is sufficient to model several of these subtleties. In AdOr-BAC, permission to delegate may be represented by facts having the following forms:

– $Permission(org, role, delegate, view, context)$

meaning that, in organization org , $role$ is permitted to delegate a permission on $view$ in a given $context$. $view$ is a sub-view of $UPA1$ or $UPA2$ (depending on the delegation is to perform a specific action on an object, or an activity on a view).

It is generally assumed that to delegate a permission to a user, the grantor must first hold the permission he wants to delegate. In AdOr-BAC, this is modelled by a context AG (for Authorized Grantor) defined as follows:

– $\forall org, \forall s, \forall a, \forall upa,$
 $Define(org, s, a, upa, AG) \leftrightarrow$
 $Use(org, upa, UPA1) \wedge$
 $Is\_permitted(s, privilege(upa), target(upa))$

that is, in any organization org , subject s performs action a in context AG if org uses upa in view $UPA1$ and s is permitted to perform the delegated privileged action on the delegated target object.

In some circumstances, we may also specify that the delegation only applies temporarily and will be automatically revoked after a given deadline. In AdOr-BAC, this may be modelled by a temporal context. Temporal and other types of contexts are further investigated in [4]. Another possible restriction is that the grantor will loose the permission he has delegated. In AdOr-BAC, this means that delegation is not an elementary activity but the combination of assigning a permission (as modelled in $UPA1$ or $UPA2$) and self-revoking this permission on the grantor (this may be also modelled in $UPA1$ or $UPA2$).

We do not further develop this analysis of the delegation concept in this paper. We plan to continue this investigation in the future.

5 Conclusion

In this paper, we have presented AdOr-BAC, an administration model for the Or-BAC model. Using AdOr-BAC, the definition of an administration policy is defined in a similar way as the remainder of the security policy specified in Or-BAC. Thus, Or-BAC is a fully auto-administered model, we suggest a logical-based model to express both Or-BAC and AdOr-BAC. In a forthcoming paper, we plan to give an interpretation of this model using a syntax closed to SQL.

AdOr-BAC provides a good compromise between fully centralized (and too rigid) administration as in the MAC model, or fully decentralized (but uncontrolled) administration as in the DAC model. When creating a new Or-BAC policy, we suggest starting with a unique user (the creator of the policy), a unique organization (whose name is defined by the creator) and a unique pre-defined role *policy-designer* assigned to the creator. The role *policy-designer* has minimal permissions to create new organizations, define roles to administer these

organizations and specify permissions associated with these roles. Thus, using AdOr-BAC, one can specify a decentralized administration, but it is always possible to control and limit the capabilities to administer associated with the different created roles.

We develop three main components for AdOr-BAC called URA for User-Role Administration, PRA for Permission-Role Administration and UPA for User-Permission Administration. The UPA component is useful to control User to User delegation, when a user wants to grant another user a specific permission. We suggest two variations of the UPA component: UPA1 that enables a user to delegate a permission to perform a specific action on a specific object and UPA2 to delegate a permission to perform an activity on a view. Applying the UPA component to model delegation still requires further work. As mentioned in [1], there are several different characteristics related to delegation such as permanence, monotonicity, totality, levels of delegation, cascading revocation. We have started modelling some of these criteria in the context of the AdOr-BAC model through context definitions. We plan to continue this work, in particular to model how to refine non elementary activities (such as non monotonic delegation) into elementary ones (such as permission assignment and self-revocation). We have not here taken to account the role hierarchy, and the inheritance cascading revocation issues which might appear then. Multi-step delegations also require further investigation. Finally, we do not discuss the enforcement of an Or-BAC security policy administrated with AdOr-BAC. This problem will be studied in a forthcoming paper.

Acknowledgement. For this work, Alexandre Miège is funded by France Télécom R&D and Frédéric Cuppens is partially funded by the MP6 RNRT project of the French ministry of Research.

References

1. Ezedin Barka and Ravi Sandhu. Framework for Role-Based Delegation Models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, New Orleans, Louisiana, December 2000.
2. J. Crampton and G. Loizon. SARBAC: A New Model for Role-Based Administration. Technical Report BBKCS-02-09, Birkbeck College, University of London, July 2002.
3. F. Cuppens, P. Balbiani, S. Benferhat, Y. Deswarte, A. Abou El Kalam, R. El Baida, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.
4. F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. In *Proceedings of 19th Applied Computer Security Associates Conference (ACSAC 2003)*, Las Vegas, Nevada, December 2003.
5. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proceedings of IEEE 2th International Workshop on Policies for Distributed Systems and Networks (POLICY 2001)*, Bristol, UK, January 2001.

6. David F. Ferraiolo and D. Richard Kuhn. Role-Based Access Controls. In Z. Ruthberg and W. Polk, editors, *Proceedings of the 15th NIST-NSA National Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.
7. S. I. Gavrilu and J. F. Barkley. Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. In *Third ACM Workshop on Role-Based Access Control*, pages 81–90, October 1996.
8. L. Guiri. A new model for role-based access control. In *Proceedings of the 11th Annual Computer Security Applications Conference*, pages 249–255, New Orleans, LA, December 1995.
9. S. Oh and R. Sandhu. A Model for Role Administration Using Organization Structure. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 155–162, Monterey, California, June 2002.
10. R. Sandhu and V. Bhamidipati. The URA97 Model for Role-Based User-Role Assignment. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*. North-Holland, Lake Tahoe, California, 1997.
11. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 Model for Role-Based Administration of Roles. *ACM Transactions on Information and System Security*, 2(1), February 1999.
12. R. Sandhu and Q. Munawer. The RRA97 Model for Role-Based Administration of Role Hierarchies. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC'98)*. Phoenix, Arizona, December 1998.
13. R. Sandhu and Q. Munawer. The ARBAC99 Model For Administration of Roles. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99)*, Phoenix, Arizona, December 1999.
14. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

Metadata Representation for Semantic-Based Multimedia Security and Management

Yuriy Rytsar, Svyatoslav Voloshynovskiy, and Thierry Pun

Computer Science Department, Stochastic Image Processing Group,
University of Geneva, 24 rue du General Dufour,
CH-1211 Geneva 4, Switzerland
{Yuriy.Rytsar, Svyatoslav.Voloshynovskyy,
Thierry.Pun}@cui.unige.ch
<http://sip.unige.ch/index.html>

Abstract. In this paper a novel approach to metadata representation for semantic-based multimedia security and management is proposed. This approach is based on semantically meaningful object segmentation and digital data hiding to facilitate image indexing as well as identification for dedicated image databases. An image is considered as a collection of the regions that correspond to objects, where these objects are associated with some hidden metadata describing some of their features. The metadata is hidden inside the raw images themselves for security and “portability” reasons. The embedding of hidden metadata allows moving an image from one database to another, as well as the insertion/cropping of objects from one image to another, while still preserving the associated descriptions. The perceptual invisibility of the integrated metadata confirms the high performance of this proposed object-based hidden metadata representation.

1 Introduction

Due to enormous increase in number of Web-based distributed environments, applications and public networks, an efficient management of digital multimedia database systems became over the last decade one of the important and challenging problems for modern multimedia applications. Taking into account the practical difficulties in controlling and verifying of existing audio/visual communications in distributed and public networks, one can imagine additional ambiguity and insecurity in extracting and managing multimedia information. Therefore, one of the possible solutions to tackle the above problem is to use the metadata. Most of Web-based applications strongly rely on metadata features for multimedia management, security and communications. Thus, in multimedia management the metadata is usually applied to facilitate proper cataloging and indexing of large amounts of visual information for effective browsing and search.

To provide complete information about the ownership of multimedia source or to describe the copyright or/and licensing information, the metadata can be easily util-

ized as well. At the same time, security issues around semantic features, annotations, metadata, object "transportability" still need to be addressed. For instance, even the simple change of file name can trick the Google image search engine and instead of the targeted data can display prohibited or age censored content. The headers of the JPEG/JPEG2K files containing metadata description can be modified exactly in the same way while the object cropping/insertion from one content to another can change semantic features, educational statements, historical or even criminal evidences of the retrieved content. The quality-of-service (QoS) control in distributed networks and particularly in heterogeneous or time-varying networks can also be provided by metadata, describing information about the channel abilities and properties. Therefore, the usage of metadata becomes an obvious and effective way to protect, manage and distribute multimedia information for current multimedia applications.

To introduce our concept of integrated metadata in different applications for multimedia management, security and communications we first define some general concepts and meanings. In the most general sense, the term of metadata means "data about data". However, in the context of images this term refers to image metadata - "data associated with an image" or "data about an image". In other words, the image metadata can be considered as all non-pixel data associated with an image or as tags, which are stored within image files [1]. In this paper, we do not follow the classical separation principle where metadata is treated independently from the raw pixel data. On the contrary, we propose to "merge" them together in a simple and elegant way using data hiding technologies.

Image metadata or data describing digital images must contain information about the image, when it was taken, by whom, with what equipment (digital or web-camera, scanner, mobile phone), copy/distribution restrictions, etc. The metadata can be generated after the image has been created to allow an event description or scene annotations. Almost all digital cameras register information about time and date of captured picture, image dimensions, etc. Since metadata becomes an integral part of the image it is necessary to provide appropriate security features against their intentional and unintentional modifications. In general even simple modification of image format will unavoidably lead to the change of metadata content (for example, from JPEG to PNG format). Unfortunately, the limited capabilities of current Web-based environment and format compatibility do not provide an adequate level of metadata attribute extraction and its sufficient protection. Therefore, new models and methods are needed for metadata protection and security while preserving efficiency.

In this paper, we present one of the possible ways to solve the above problems of metadata efficiency and security by the embedding of image annotations inside the raw image data. Thus, in digital database the images will be indexed by their own visual content instead of being annotated by text-based keywords and the metadata will be completely hidden for non-authorized access. Moreover, this approach will provide more security for image information as well as the image "portability" from one digital database to another without additional associated descriptions.

This paper is organized as follows. In Section 2 we describe interactive image segmentation for content-based image extraction. The object-based metadata hiding approach is described in Section 3. In Section 4 we demonstrate experimental results

of region-based segmentation with metadata hiding. Section 5 provides some concluding remarks.

2 Interactive Segmentation for Content-Based Image Extraction

The main difficulty of existing image and video database indexing and searching systems [2-5] is that they cannot automatically describe multimedia content using both low-level (color, shape, texture) and semantic (objects, people, places) descriptors [6]. Usually, the researchers utilize various complex visual features like color distribution, object shapes, textured surface or simple textual descriptions like annotations, keywords, queries or captions to search in large databases. Besides, the usage of the semantic features requires a good correspondence between the description and the content of multimedia data. Another problem arises when an object is cropped or inserted from one image into another image. In this case the integrated descriptions will be completely lost due to the fact that metadata is separated from image data.

The other difficulty is that the same image content may be interpreted by different people in completely different ways. Subjective human descriptions may cause ambiguity and mismatches in the extraction process and in the management of multimedia information. The relevance of visual content significantly depends on the subjectivity of both a database provider and a user. Although a lot of study and publications are dedicated to this area and many retrieving systems are already developed, there are still many open research and commercial issues to be solved and applied in practice [7-10].

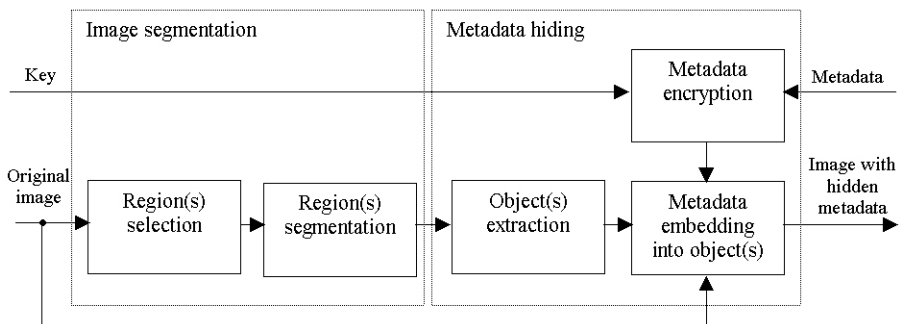


Fig. 1. Block diagram of metadata embedding

We propose the following content object-based retrieval approach as a possible solution to the above mentioned problems. It is based on a human-computer interface for semantics extraction and object-based metadata integration by using digital data hiding technologies. Instead of being annotated by text-based keywords, the images may be labeled by their own semantic content, which is hidden into every image object. Since each homogeneous region in the image has common features (brightness, color distribution, textured surface) it is possible to exploit this fact for an ob-

ject-based extraction by performing semantic object segmentation. Different objects can be semantically distinguished regarding their own features and easily identified by a search engine. The problem of classical unsupervised segmentation is its inability to capture semantic relationships in real world images. That is why human assistance and feedback are needed to guide the process of semantically meaningful segmentation in order to make an adequate correspondence of each object of the image to its integrated metadata content.

In this paper we do not focus on specific segmentation algorithms (for example, [11-14]), we rather refer to the fact that successful multimedia retrieval critically depends on the chosen segmentation technique based on region or object content. The successful segmentation of the image is heavily dependent on the criteria used for the merging of pixels based on the similarity of their features, and on the reliability with which these features are extracted. An unsupervised stochastic segmentation is used as a first iteration and the result is displayed for the user. The user then defines where the region of interest is, and the chosen segmentation algorithm performs partitioning into meaningful objects. The human-computer interface is organized in such a way that the user can easily add/remove some objects or parts of objects from previous stages, select objects of interest, or even merge some objects that are classified as distinct according to their statistical properties but in fact represent parts of the same semantically connected object.

Hence, the image is not considered as a set of pixels but rather as a set of annotated regions that correspond to objects or parts of objects, where these objects are associated with some hidden descriptive text about their features. Each selected object of the image carries its own embedded description that makes it self-containing and formally independent from the particular image format used for storage in image databases. Therefore, the proposed approach of metadata representation can easily provide “portability” of each object from one image to another, as well as makes resistant to object insertion or/and cropping, which are usually used in multimedia processing and management. Moreover, the image can be moved from one database to another without any associated descriptions because of the self-containing features of objects. One can consider this feature as a joint distribution of visual and textual description (image and text).

The block diagram of the proposed approach for metadata embedding into image objects is shown in Figure 1. First, the user selects region(s) of interest on a given image. Region-based segmentation is then applied for pre-selected region(s) in order to extract object(s) of interest for their further description. Secondly, the user embeds the previously encrypted metadata (text description or short annotation) inside the selected object(s) based on robust digital data hiding technique and personalized secret key.

3 Object-Based Metadata Hiding

The second part of the proposed approach is based on the hidden indexing/labeling of image objects by using robust digital data hiding technique. Here, we are focused

neither on the process of metadata encryption and hiding (see more in [15]) nor on the robustness of data hiding algorithms against some attacks; this is out of the scope of this paper. We rather discuss the possible ways where and how to integrate the metadata reliably into the image and then, how to extract it without errors. We intend to embed the semantically segmented map as well as the user assigned description into the body of each object as the robust watermark. Thus, the image is considered as a set of "smart objects" divided by boundaries that can perform:

- self-indexing;
- self-authentication;
- self-synchronization;
- self-extraction;
- self-tamper proofing.

In other words, the image contains all necessary information about itself including descriptions of the object bodies, of their mutual allocation in images, as well as complementary hidden metadata. Moreover, no additional header, attachment, tag or extra metadata are needed for further image indexing and identification.

It should be noted that the metadata embedding/extraction could be based on symmetric or asymmetric protocol en/decryption. The *symmetric protocol* means that only the authorized party can embed, modify and retrieve the hidden information by using a private secret key. The *asymmetric protocol* means that only the authorized party can embed and modify the hidden metadata by using a private secret key, and that the rest of users can only extract it by using a public key. In both cases the additional security level of metadata is provided. Consider one possible example of the proposed system for telemedicine application: for confidentiality reasons it is based on a symmetric protocol. Each region of interest has its own label with hidden description connected to the extracted object, and the private secret key is used for information embedding. For example, a physician working with the MRI of a patient after having used the interactive segmentation tool for the detection of possible tumor or clot of blood will be able to insert the necessary information directly into the image or into some parts of this image. Even if the patient or any unauthorized party recovers his MRI data he will be unable to extract this description without secret key. However, the patient can give this image to another physician, who will be able to directly extract this invisible hidden information using the same segmentation tool and the proper private key. Therefore, time and money will be saved as well as security and confidentiality of this protocol will be provided.

As a possible extension, the semantic content of the object body can be replaced by other hidden multimodal media data (see more in [15]). For example, each object may have its own audio/musical context and this feature may be used for blind people (special interfaces, web-browsers). Another possible extension would consist in encryption or steganographical applications, meaning the hiding of small amounts of textual or visual raw-data into selected "smart objects". Besides, the raw-data about the contour of the object body can be additionally embedded into the same object. In other words, the image has more information about itself and this information is perceptually invisible and is not integrated inside image headers.

The block diagram for metadata extraction from the image objects is shown in Figure 2. First, the user selects the region(s) of interest on a given image. Region-based segmentation is then applied for the pre-selected region(s) in order to extract object(s) of interest to obtain their descriptions. Secondly, the user retrieves the metadata from the extracted object(s) based on the secret key.

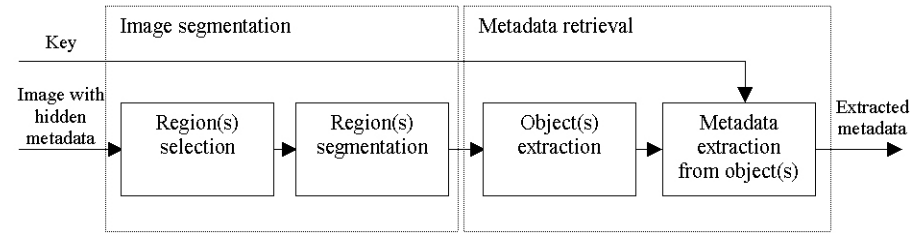


Fig. 2. Block diagram of metadata extraction

Obviously, this visual and textual representation of the image needs additional efforts to keep all information more secure and robust to possible modifications and transformations. Besides, the synchronization of the resulting segmentation process is required for both metadata embedding and extraction. After reliable and successive application of the segmentation technique only an authorized party can extract object(s) of the image to retrieve the hidden metadata bits associated with the given object(s) by using a secret private key. The invariance to different types of distortions can be achieved by providing additional embedding of hidden information into image objects, robust to such changes. Therefore, for robustness reasons the object descriptions can be additionally encoded by error correction codes (ECC) (Turbo code [16] or LDPC code [17], for example) depending on the applications.

In order to avoid possible multiple insertion of contradictory metadata into the same image object we consider several scenarios, which can be applied for the metadata embedding. As the first scenario, we propose to perform preliminary detection before the insertion of metadata information into selected image object. If the preliminary metadata extraction is successful no additional information will be embedded into the object. Another possible scenario is complete metadata replacement, where the previously embedded information related to the object is completely replaced by new data by the user request. In the case, when it is necessary to add complementary information to the embedded metadata we propose to check the “free space” for the location of an additional raw pixel data. If this space is enough to perform the additional embedding, one can also replace completely the metadata information or leave it unchanged depending on the scenario.

4 Application Scenarios

The performance of the presented technique was tested for embedding and extraction of the metadata for image objects on different grayscale and color images. Here are considered two application scenarios of the proposed approach for visual annotations of the images and for image indexing and search in databases. In Figure 3 the experimental results of metadata hiding for medical MRI image of axial slice of the human femur are shown. The region of interest is defined and marked by using a human-computer interface. The corresponding textual information about the selected image object (cortical bone and marrow) is encrypted based on the secret key, and is then embedded into marked image regions. One can find that the image quality with embedded metadata is not perceptually degraded. The PSNR value is equal to 38 dB.

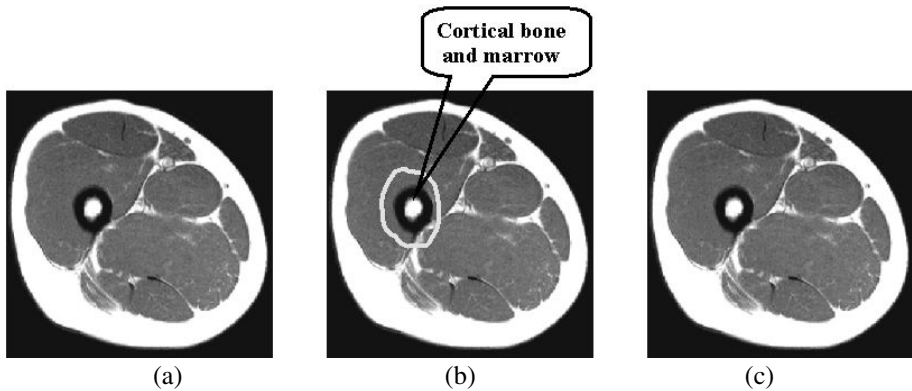


Fig. 3. Integrated metadata for medical MRI image: (a) original medical MRI image of axial slice of the human femur; (b) visually annotated image with marked region of interest; (c) the resulting image with hidden visual annotation

For the second application scenario, we used several images from an official Formula 1 car racing website [18] (Figure 4a, 5a). We performed local unsupervised region-based segmentation in order to define regions of interest for the original image. After, we selected and merged together small segmented objects into one semantically meaningful object by using the human-computer interface (Figure 4b). Then, the marked object was labeled by a corresponding textual description and integrated into the object by using a symmetric encryption procedure (Figure 4c). This metadata was used for further indexing and search for content-based databases.

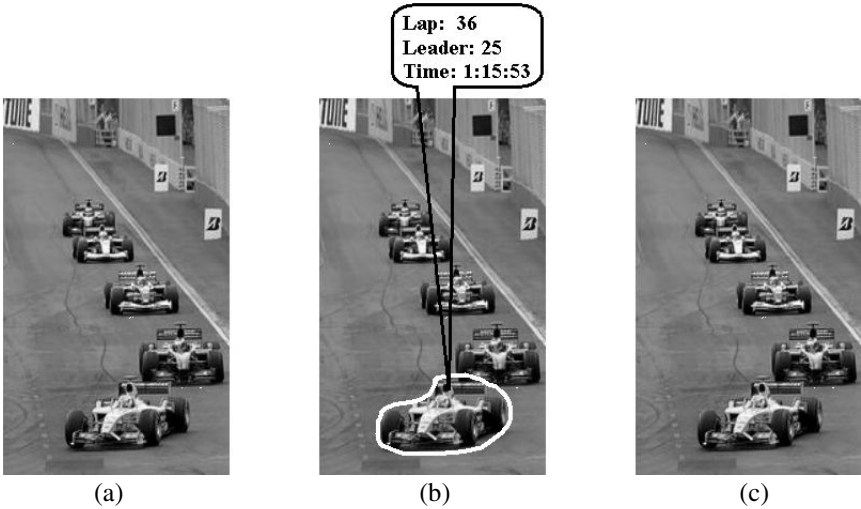


Fig. 4. Integrated metadata for the car racing image: (a) original image; (b) image with marked object; (c) the resulting image with hidden description

We copied and inserted the marked object (yellow car) into another test image (see Figure 5b). Since the metadata was integrated and hidden into image objects, the description information was simultaneously “transported” with this object to others images, while image quality was completely preserved (see Figure 5c). During the retrieval stage all hidden metadata was successfully extracted from this test image based on the same secret key.

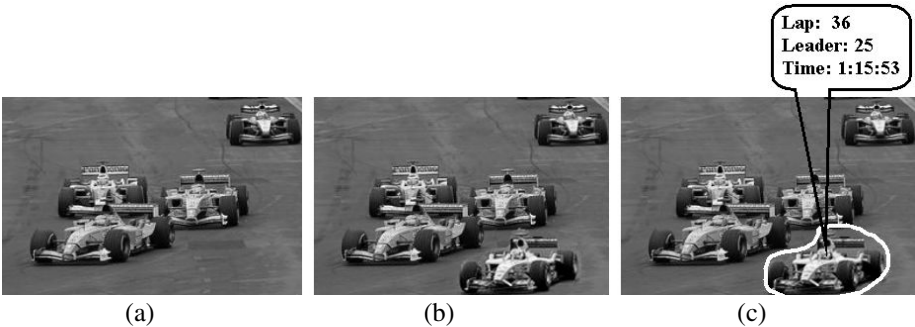


Fig. 5. Extracted hidden metadata for the target image: (a) target image; (b) image with inserted object; (c) the resulting image with extracted hidden description

Since the identical object is present in both images, the only possible way to distinguish and retrieve them from an image database is based on the proper hidden metadata of this object, but not on the visual data itself. We mean the corresponding textual description such as general information about object, its identification number, date and time of image creation, ownership, spatial size of given object, short information about the image to which this object belongs.

Here we are not focused on copyright protection, but we rather demonstrate the ease with which the object information can be kept reliably and properly. Obviously, the retrieved metadata from those images can be used to prove the origin of the photographs, as well as to outline the modifications that have been performed on them (see more details about the tamper proofing and authentication techniques in [19, 20]).

It should be noticed that for both application scenarios the image quality with embedded metadata is not perceptually degraded. Besides, on the second example we demonstrated the “portability” features of our technique where one image object can be easily inserted into another image without visible degradation and loss of embedded information.

5 Conclusion

In this paper a hidden metadata representation approach has been proposed for content management and reliable communications. The presented technique applies interactive object-based segmentation for metadata hiding into extracted objects of the image in order to provide additional protection and security. Moreover, the embedded metadata is independent from the particular format used for storage in an image database, and is robust to any format changes as well as to insertion/cropping image transformations. Besides, the perceptual invisibility of the integrated metadata confirms the high performance of proposed approach.

Acknowledgment. This work was partially supported by the Swiss National Center of Competence IM2 – Interactive Multimedia Information Management.

References

1. Milch, J.: Eastman Kodak Company. Hints for the Creation and Usage of Picture Metadata (2000)
2. Smith, J.R., Chang, S.-F.: VisualSEEK: A fully automated content-based image query system. *Proc. ACM Multimedia'96* (1996)
3. Ogle, V.E., Stonebraker, M.: Chabot: Retrieval from relational database of images. *Computer*, Vol. 28. 9 (1995) 40–48
4. Müller, H., Müller, W., Marchand-Maillet, S., Squire, D. McG., Pun, T.: A Web-Based Evaluation System for Content-Based Image Retrieval. *Proceedings of the Multimedia Workshop on Multimedia Information Retrieval*, Ottawa, Canada (2001)
5. Gevers, T., Smeulders, A.W.M.: Pictoseek: combining color and shape invariant features for image retrieval. *IEEE Trans. on Image Processing*, Vol. 9. 1 (2000) 102–119
6. Xu, Y., Saber, E., Tekalp, A.M.: Object Segmentation and Labeling by Learning from Examples. *IEEE Trans. on Image Processing*, Vol. 12. 6 (2003) 627–638

7. Damiani, E., De Capitani di Vimercati, S., Fernández-Medina, E., Samarati, P.: An Access Control System for SVG Documents. Proc. of the Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security, King's College, University of Cambridge, UK (2002)
8. Müller, H., Müller, W., Squire, D. McG., Marchand-Maillet, S., Pun, T.: Performance Evaluation in Content-Based Image Retrieval: Overview and Proposals. Pattern Recognition Letters, Special Issue on Image and Video Indexing. (eds.): H. Bunke and X. Jiang, Vol. 22. 5 (2001) 593–601
9. Rui, Y., Huang, T. S., Chang, S.-F.: Image retrieval: Past present and future. J. of Visual Communication and Image Representation, Vol. 10. (1999) 1–23
10. Boulgouris, N.V., Kompatsiaris, I., Mezaris, V., Simitopoulos, D., Strintzis, M.G.: Segmentation and Content-based Watermarking for Color Image and Image Region Indexing and Retrieval, EURASIP Journal on Applied Signal Processing, 4 (2002) 420–433
11. Zhang, Y.J.: Evaluation and Comparison of Different Segmentation Algorithms. Pattern Recognition Letters, Vol. 18. 10 (1997) 963–974
12. Pal, N.R., Pal, S.K.: A review on image segmentation techniques. Pattern Recognition, Vol. 26. 9 (1994) 1277–1294
13. Kurugöllü, F., Sankur, B., Harmanci, E.: Multiband Image Segmentation Using Histogram Multithresholding and Fusion. J. of Image and Vision Computing, Vol. 19. 13 (2001) 915–928
14. Duygulu, P., Vural, F.: Multi-Level Image Segmentation and Object Representation for Content Based Image Retrieval. SPIE Electronic Imaging 2001, Storage and Retrieval for Media Databases. San Jose, CA (2001)
15. Rytsar, Y., Voloshynovskiy, S., Ehrler F., Pun, T.: Interactive Segmentation with Hidden Object-Based Annotations: Toward Smart Media. SPIE Electronic Imaging 2004, Storage and Retrieval Methods and Applications for Multimedia. San Jose, CA (2004) (*accepted*)
16. <http://www331.jpl.nasa.gov/public/JPLtcodes.html>
17. Gallager, R.G.: Low Density Parity Check Codes. Monograph, M.I.T. Press (1963) (<http://www.inference.phy.cam.ac.uk/mackay/gallager/papers/ldpc.pdf>)
18. Australian Grand Prix 2003 postcards: <http://www.formula1.com/gallery/images/2/Sunday/1949.html>
19. Voloshynovskiy, S., Deguillaume, F., Koval O., Pun, T.: Robust digital watermarking with channel state estimation: part II Applied robust watermarking. Signal Processing (2003) (*submitted*)
20. Deguillaume, F., Voloshynovskiy, S., Pun, T.: Secure hybrid robust watermarking resistant against tampering and copy-attack. Signal Processing (2003) (*submitted*)

Comparing Logic-Based and XML-Based Rights Expression Languages

Cheun Ngen Chong, Sandro Etalle, and Pieter H. Hartel

Faculty of EEMCS
University of Twente
The Netherlands.

{chong,etalles,pieter}@cs.utwente.nl

Abstract. Several rights expression languages (RELs) have been proposed to describe licenses governing the terms and conditions of content access. In this field XrML and ODRL play a prominent role. Both languages are powerful yet complex. In this paper we propose a way of analysing RELs and we apply it to ODRL, XrML and to LicenseScript, a REL we propose. In addition, we test these languages against a number of example scenarios. These examples bring new insights, and shed new light on some of the limits of XrML and ODRL.

1 Introduction

Right expression languages (RELs) are languages devised specifically to express the condition of use of digital content in general, and of multimedia in particular. RELs can for instance be used to describe an agreement between a content provider and an music distributor, or to express the copyright associated to a given piece of music, by specifying under which conditions the user is allowed to play, broadcast, or copy it.

In the vast scene of multimedia delivery, two RELs in particular have attained a prominent position: XrML [7] and ODRL [8]. The eXtensible rights Markup Language (XrML) (<http://www.xrml.org>) is proposed and maintained by ContentGuard, Inc. (<http://www.contentguard.com>), and has its roots in the Stefik's Digital Property Rights Language. XrML is adopted by Microsoft in Windows Media Player 9. The Open Digital Rights Language (ODRL) (<http://www.odrl.net>) was proposed by Iannella from IPR Systems Ltd. (<http://www.iprsystems.com>). ODRL is endorsed by the Open Mobile Alliance (OMA) (<http://www.openmobilealliance.org>).

XrML and ODRL have many similarities: syntactically they are both based on XML while structurally they both conform to the Stefik's axiomatic principles of rights modelling (<http://www.oasis-open.org/cover/DPRLmanual-XML2.html>).

XML-based RELs, however, have some intrinsic disadvantages: (1) the syntax is complicated and obscure when the conditions of use become complex, (2) these languages lack a formal semantics [11,6], i.e. the meaning of licenses relies heavily on human interpretation, and (3) the languages cannot express many useful copyright laws [9]. To address these problems we have proposed a new, logic-based REL, named LicenseScript [3]. LicenseScript has a declarative as well a procedural reading (i.e., can be used as a programming language), which makes it possible to capture a multitude of sophisticated usage patterns precisely and unambiguously.

LicenseScript provides an approach to RELs which is diametrically opposite to that of XrML and ODRL: it is logic-based rather than XML-based. This makes it difficult to make an objective assessment and comparison of the two REL styles. Such an objective assessment is important for a clear understanding of the advantages and the limitation of XrML and ODRL. Last but not least, it must be noted that making such assessment is far from trivial, as ODRL and XrML specifications are huge and complex.

This paper aims (at least partially) at solving the aforementioned problem (i.e. complexity of XML-based RELs). Our contribution is twofold: first, we develop an anatomy of the RELs, then we apply it to ODRL, XrML and LicenseScript; secondly, we analyse in depth a number of examples, which we have coded in LicenseScript as well as in ODRL and XrML. In our opinion, these examples bring new insights, and shed new light on some of the weaknesses of XrML and ODRL.

The organization of the remainder paper: Section 2 discusses the anatomy of the RELs. Section 3 describes briefly the LicenseScript language to make the paper self-contained. Section 4 discuss the results of our studies on the scenarios specified in XML-based RELs. Section 5 describes some of the novel scenarios in LicenseScript. Lastly, section 6 concludes the paper and presents future work.

Note: Due to space constraints, we are forced to put our studies of XML-based RELs scenarios in our Technical Report version of this paper [4]. The appendix in the Technical Report lists the LicenseScript code of the XML-based RELs scenarios.

2 Anatomy of RELs

To aid the comparison of RELs we propose an anatomy of the RELs. Based on Stefik's axiomatic principles, the XrML and ODRL specifications, and the requirements of RELs proposed by Parrott [10], we conclude that RELs have a structure which is shown in Fig. 1. The figure is presented in the form of a class diagram because this exhibits the logical relations between the components. This figure provides an abstract view of a REL.

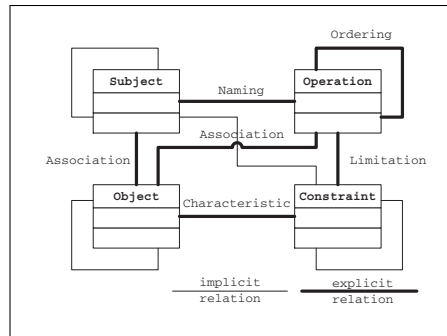


Fig. 1. The components and their relations in a REL.

We identify four main components, namely *subject*, *object*, *operation*, and *constraint*. We explain these components in section 2.1. Each of these components is logically related to other components. We elaborate these *relations* in section 2.2. The components and the relations established within support a wide variety of *models* of the rights management systems. We elaborate the models in section 2.3. Most what follows come from the existing material (Parrott [10] and XML-based RELs specification), but we explicitly indicate the additional features of a REL.

2.1 Components

From Parrott's requirements of RELs [10] and the XML-based RELs specification, we conclude that there are four main components in a REL, namely (1) *subject*, which is an actor who performs some operations; (2) *object*, which is the content acted upon by a subject; (3) *operation*, which is what a subject can do to an object; and (4) *constraint*, which describes when an operation can be performed. There are two types of operation: a *right* is an operation that can be performed directly on the object; and an *obligation* is an operation that must precede or follow another operation. As an illustration consider the following:

Example 1. Alice wants to play a high-quality movie three times, she must pay \$5 upfront.

Intuitively, the subject is "Alice", the objects are "movie" and "\$5", the right is "play", the obligation is "pay", and the constraints are "high-quality", "for three times" and "upfront".

A REL typically describes the *subject* by naming, e.g. "Alice". Additionally, the REL must be able to distinguish the type of the subject by using the names, e.g. creator, end-user, distributor and so on. A REL must also be able to specify the identification mechanism employed by common rights management systems to describe the subject, e.g. a digital certificate and public key. This is discussed in section 2.3.

A REL uses names to describe an *object*, such as the title of the object or the artist. A REL is required to support (1) generalized types of objects, for instance, multimedia (e.g. MP3), personal data (e.g. DOC) or meta-data (e.g. XML); (2) classification of similar objects, for instance, "publisher Addison's ebooks"; (3) fuzzy (or implicit) matching criteria of the object, for instance, "looks like" and "sounds like"; and (4) the delivery methods of the object, for example, by downloading, streaming or by means of physical storage (e.g. CD).

As mentioned earlier, there are two types of *operation*: *right* and *obligation*. There are various types of rights identified by Rosenblatt et al [12]: (1) *Render*, which indicates a set of rights in which the object can be consumed, e.g. play; (2) *Reuse*, which indicates a set of rights in which the object can be re-utilized, e.g. modify. (3) *Transport*, which indicates a set of rights in which the subject's rights over the object can be transferred, e.g. lend. (4) *Object management*, which indicates a set of rights to handle the management over the object, e.g. move and duplicate. None of these rights cover the regulation of the rights themselves. Therefore, in addition, we propose a further set of rights, namely (5) *Rights regulation*, which indicates a set of rights that regulate the subject's rights over the object, e.g. update and renew.

A REL should be capable of describing different obligations. An obligation may be an operation that enables or activates the rights over the objects. The *pay* and *register* operations are two common examples.

Parrott [10] and XML-based RELs specifications recognize several common *constraints*: (1) *temporal*, such as date and time (e.g. the ebook can be viewed before 20 March 2004); accumulated (e.g. the ebook can be viewed for 2 weeks); and interval (e.g. the ebook can be viewed within 20 days from the time of issuing this license); (2) *bound*, for instance, the number of distinct times the ebook can be viewed, and the

range of the page numbers of the ebook that can be printed; (3) *environment*, which may be a physical environment (e.g. geographic territory) or logical environment (e.g. network address or system environment); (4) *aspect*, which mainly relates to the technical perspectives of the object, for example, quality and format of the content; (5) *purpose*, for instance, educational purpose and commercial reason. There may be more unique constraints required when new scenarios emerge.

We introduce another constraint, namely the *status* constraint. Real time content access requires this constraint to indicate the current state, e.g. availability and accessibility of the content at the time the rights are exercised.

XrML and ODRL are able to represent render, reuse, transport, and object management rights. However, XrML and ODRL do not accommodate (explicitly) the descriptions of rights that regulate other rights. For example, “a user can *renew* the rights to play a movie within a fixed period (after the expiry time of the rights) with a discount”. However, XrML and ODRL do cater for the revocation of rights and obligations. XrML does not provide explicit facilities to specify the purpose constraints. ODRL and XrML cannot express the status of the object. However, LicenseScript is able to accommodate most (if not all) the listed constraints.

2.2 Relations

A REL must specify *relations* between components. As can be seen in Fig. 1, there are two distinct types of relations, namely *explicit* relations and *implicit* relations. We use example 1 to elaborate some of the relations discussed in this section.

Parrott [10] identifies two classes of explicit relations, namely *ordering* relation, e.g. “pay \$5 before *play* the movie” (operation–operation); and *association* relation, “Alice owns the *movie*” (subject–object) and “*play* is for *movie*” (operation–object). The *ordering* relation describes how operations are linked. For example, “pay before play” is an example of antecedent obligation; and “play then pay” is an example of consequent obligation. An ordering can be total or partial. A total ordering fully specifies the order of all operations, for example, “register, pay and then play”. A partial ordering implies there is no explicit order between all items, for example, “register and then play, user can pay before or after”. The *association* relation covers the subject–object and operation–object relations.

We identify three additional types of explicit relations, namely, (1) *naming* relation (subject–operation), which specifies the name of the operation the subject can perform, e.g. “Alice *plays* the music”; (2) *limitation* relation, which implies that the operations are restricted by the constraints, in the same example, (constraint–operation); and (3) *characteristic* relation (constraint–object), which describes the object (that the operations can be acted upon), e.g. “*high-quality movie*”.

We also identify several implicit relations (see Fig. 1), which include: subject–subject, subject–constraint, object–object, and constraint–constraint. These implicit relations are embedded and indirect. To elaborate these relations, we use two additional examples:

Example 2. Alice needs Bob to prove her identity so that she can play the movie.

Example 3. Alice can reuse the image in the ebook on her Web site, for educational purpose and for 2 years.

Example 2 exhibits the implicit subject–subject relation between “Alice” and “Bob”, as well as implicit subject–constraint between “Alice” and “prove her identity”. Example 3 exhibits implicitly the object–object relations between the “image” and “ebook”, and the constraint–constraint relations between the “educational purpose” and “2 years”.

2.3 Models

A *model* describes a typical way of using a REL; we can distinguish: (1) *revenue* model, (2) *provision* model, (3) *operational* model, (4) *contract* model, (5) *copyright* model and (6) *security* model. A rights management system may exhibit different models simultaneously.

The *revenue* model, is normally related to the payment architecture of the system. There are myriad of revenue models, for example, pay-per-use, pay-upfront, pay-flatrate, tiered payment (e.g. free now pay later), pay to multi-entities (e.g. pay half to publisher and half to distributor), and fraction payment (e.g. discount and tax). New revenue models emerge every day.

The *provision* model may provide an *alternative solution* more than yes or no to the situations when the rights and obligations fail to meet the constraints. For instance, if viewing a high-resolution video is not allowed, it should be possible to switch to low-resolution video. Additionally, the provision model should be able to *reconcile the conflicts* caused, for example, when there is more than one subject performing the same operation on the same object simultaneously. The provision model also accommodates the *default settings of operations* over an object when the object is not associated with any operations.

The *security* model defines a variety of security mechanisms, for instance, identification, authentication and authorization (IAA), access control, non-repudiation, integrity, audit trails and privacy.

The *operational* model handles the technological aspects of the system, such as quality-of-service, watermarking, caching, network operations, bandwidth and other operational aspects of the system.

The *contract* model establishes the agreement of the terms and conditions (over the operations offered over the object and constraints) established between different subjects.

We include the *copyright* model in this category because the copyright enforcement from the user’s standpoint is always a source of controversy [2]. The *copyright* model enforces copyright acts (especially from the end-user’s standpoint), such as fair use, first sale and so on.

Not all RELs are able to support the 6 revenue models above. XrML and ODRL are not able to support the provision model of reconciling the rights conflicts. This model handles the dynamic license evolutions and content access patterns. XrML and ODRL are static RELs that are not sufficiently flexible to meet this requirement.

None of the RELs can as yet support the copyright model [13]. However, Mulligan and Burstein [9] provide several suggestions to incorporate copyright into the XML-based RELs. We address this issue as our future work.

In the following section, we describe concisely the LicenseScript language using a simple scenario as an example.

3 LicenseScript Language

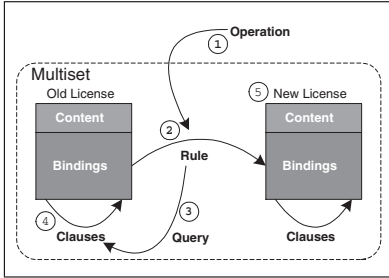


Fig. 2. Transformation of licenses with content and bindings caused by rules.

LicenseScript is a language that is based on (1) multiset rewriting, which captures the dynamic evolution of licenses; and (2) logic programming, which captures the static terms and conditions on a license. LicenseScript provides a judicious choice of the interfacing mechanism between the static and dynamic domains.

A *license* specifies when certain operations on the object are permitted or denied. The license is associated with the *content*, as can be seen in Fig. 2. A license

carries *bindings*, which describe the attributes of the license; and the *clauses*, which determine if a certain operation is allowed (or forbidden). The license clauses consult the license bindings for their decision making and may also alter the values of the license bindings.

Licenses are bound to the terms that reside in multisets. For the specification of the licenses, we use logic programming. The readers are thus assumed to be familiar with the terminology and the basic results of the semantics of logic programs.

Fig. 2 illustrates that (1) an *operation* (performed by a subject) (2) invokes a *rule*, which in turns picks the required license in the multiset. The rule then generates and executes a (3) *query* on the license (4) clauses. The (5) execution result of the rule is a newly generated license. We elaborate this transformation process later by using a simple scenario.

Now we use a simple illustrative scenario to explain the LicenseScript:

Example 4. Amanda gets an ebook, titled “A Book” from Ben Publisher. Ben issues a license with an expiry date fixed at “23/06/2004”.

This license allows Amanda to print two copies of the ebook (L01, . . . , L14 are line numbers included for reference purposes, they are not part of the code):

```

license(ebook:a_book,                                L01
[   (canprint(B1,B2,User) :-                          L02
    get_value(B1,consumer,C),                          L03
    C = User,                                           L04
    get_value(B1,expires,Exp),                          L05
    today(D), D>Exp,                                   L06
    get_value(B1,printed,P),                            L07
    get_value(B1,max_prints,Max),                        L08
    P < Max,                                             L09
    set_value(B1,printed,P+1,B2)],                     L10
[   (company=ben_publisher),                           L11

```

```
(consumer=amanda),           L12
(expires=23/06/2004),         L13
(max_prints=2), (printed=0)   ] ) L14
```

A license is represented by a term of the form `license(content,C,B)`, where `content` is a unique identifier referring to the object of the license; `C` is a list of *license clauses* (i.e. Prolog programs) describing under which circumstances the operations are permitted or denied; and `B` is a list of *license bindings* capturing the attributes of the license. We define two multiset-rewrite *rules*, as shown below, to model the interface between the system and the licenses. The rules can be thought of as a firmware in the user's system.

The syntax of the rules is based on the Gamma notation [1] of multiset rewriting (again, R01, . . . , R04 are line numbers):

```
print(Ebook,User) :           R01
    license(Ebook,C,B1) ->    R02
    license(Ebook,C,B2)       R03
<= C |- canprint(B1,B2,User)  R04
```

We will step through the example assuming Amanda would like to print the eBook with the available license on her system:

1. Amanda's system wants to know whether she has the print *right* on the eBook. This is achieved by applying the *print rule* (line R01) with appropriate parameters: `print(ebook:a_book,amanda)`.
2. The rule finds the `license(ebook:a_book,[...],[...])` (line R02, line L01) in the system. The first list refers to the license clauses (lines L02–L14), while the second list refers to the set of license bindings (lines L15–L18).
3. The rule then executes a *query* in the form of `canprint(B1,B2,User)` (line R04), where `B2` designates the output generated by the query; This will form a new set of license bindings.
4. The license interpreter retrieves the value of the license binding `consumer` from the list of license bindings `B1` (line L03) and compares the retrieved value with the user identity `User` (line L04). `User` is passed in as an argument to the license clause (line L02).
5. Similarly, the interpreter retrieves the value of the binding `expires` (line L05).
6. The interpreter calls the primitive (which is discussed later) `today(D)` (line L06) to obtain the current time and date.
7. The expiry date of the license must be greater than current time and date (line L07).
8. Similarly, the value of `printed` is checked if it is smaller than the value of `max_prints` (lines L07–L09).
9. If all conditions are satisfied (the user is valid, the license has not expired and the number of printed copies does not exceed the allowable maximum copies), the query returns *yes* (line R04) to the interpreter, with the newly generated license bindings in `B2`.
10. The value of the license binding `printed` is incremented (line L10) every time the print operation succeeds.
11. The value *yes* indicates that the execution of `canprint(B1,B2,User)` yields *success* in the license clauses `C`.
12. The rule `print(Ebook,User)` generates a new license with the newly generated license bindings `license(Ebook,C,B2)` (line R03).

The primitive `get_value(B, n, V)` is to report in V the value of n from B ; whereas the primitive `set_value(B1, n, V, B2)` is to give value V to n in $B2$. In addition, we use a number of primitives to model the interface of the system with the license (interpreter), for instance: (1) `today(D)`, to bind D to the current system date/time; and (2) `identify(L)`, to identify the current environment to L . For further details of the LicenseScript language, see Reference [3].

4 XML-Based RELs Scenarios

As mentioned in section 1, LicenseScript is intrinsically distinct from XrML and ODRL. To help comparing them, we have studied 20 scenarios from the XML-based RELs documentations.

Table 1 summarizes the analysis of all the scenarios that we have studied. The rows of the table correspond to the anatomy presented in section 2. The columns correspond to the scenarios, which are:

- ODRL scenarios, which are specified in ODRL specification (<http://odrl.net/1.1/ODRL-11.pdf>): ebook #1 (E1), #2 (E2), #3 (E3), video (VD), superdistribution (SD), software(SW), image(IM) and audio(AD);
- XrML scenarios, which are specified in XrML Use Case Examples (<http://www.xrml.org/spec/2001/11/ExampleUseCases.htm>): preview/ promotional (PP), subscription (SB), territory restriction (TR), temporal ordering of rights (TO), usage of part of a work (PW), site license (SL), personal lending (PL) and giving (GV), superdistribution (SD), unrestricted sales (US), personal copies (PC), web service access (WS), software execution (SW), confidentiality of rights (CR), operational model (OM) and secure device (SV).
- The last two columns refer to two novel scenarios, namely project documents sharing (PS) and license evolution modelling (LM), which we will discuss later.

We have translated the scenarios listed in Table 1 in LicenseScript. Thereby, we put a ‘✓’ where we can show that LicenseScript covers the corresponding feature of the REL. However, due to space constraints, we have put the LicenseScript code in the technical report [4] version of this paper.

Here, we list the conclusions from conducting the analysis of XML-based RELs scenarios:

1. LicenseScript can render most of the aspects of XML-based RELs, for instance, payment, user authentication etc., distinct and explicit by using the primitives and the license clauses. We use user authentication as an example: in XML-based RELs, the user identity (e.g. certificate, public key, etc.) is described by using XML tags. In LicenseScript, we use license binding to carry the user identity and use (in)equalities in the license clauses to model the user authentication explicitly.
2. LicenseScript can capture the dynamic license (along with the content) transformation caused by the operations by using the rules. For instance, we can model the transformation of an offer to a license, when the user *agrees* on the terms and conditions stated in offer (see section 5.2). We can also clearly capture the behaviour of the license evolutions when the license is copied, clipped or mixed, etc.

3. LicenseScript can model a wide variety of distinct terms in a multiset, which could be a license, an offer, a wallet, a policy, etc. This offers a high flexibility in designing systems that are not confined to rights management. For instance, we can model a flexible payment system by using LicenseScript [5].
4. LicenseScript can provide a fine granularity of control over the content, as do XML-based RELs, as shown in Table 1. The table also shows that that LicenseScript is more flexible and expressive than XML-based RELs.

In subsequent sections, we use LicenseScript to describe the aforementioned novel scenarios. Thereby, we show some of the advantages of LicenseScript over XML-based RELs in addition to the listed conclusions.

5 Novel Scenarios

In this section, we use two novel scenarios to demonstrate the advantages of LicenseScript over the XML-based RELs.

5.1 Project Document Sharing

Fred, his project teammate Greg, and his project leader Han use a document sharing system for collaboration. Sometimes conflicts rise, for example, when Fred and Han are working on the same document at the same time. To reconcile these conflicts, they agree on a policy that Han's version always overwrites others' versions. Ian (from outside of the institute) has joined the project recently. Han has the privilege to grant him the system access.

The license for document "State-of-the-Art.tex", which is created by Fred, can be written as follow:

```
license(tex:state-of-the-art.tex,
[ (can_startread(B1,B2,User) :-
    get_value(B1,members,Ms), member(User,Ms), get_value(B1,systems,SYSSs),
    identify(D), member(D,SYSSs), get_value(B1,isreading,Rs), not(member(User,Rs)),
    append(User,Rs), set_value(B1,isreading,Rs,B2), get_value(B1,history,H),
    today(D), append([User,startread,D],H,H2), set_value(B1,history,H2,B2)),
  (can_endread(B1,B2,User) :-
    get_value(B1,isreading,Rs), remove(Rs,User,NR), set_value(B1,isreading,NR,B2),
    get_value(B1,history,H), today(D), append([User,endread,D],H,H2),
    set_value(B1,history,H2,B2)),
  (can_startwrite(B1,B2,User) :-
    get_value(B1,members,Ms), member(User,Ms), get_value(B1,systems,SYSSs),
    identify(D), member(D,SYSSs), get_value(B1,iswriting,Rs), length(Rs)=0,
    append(User,Rs), set_value(B1,iswriting,Rs,B2), get_value(B1,history,H),
    today(D), append([User,startwrite,D],H,H2), set_value(B1,history,H2,B2)),
  (can_startwrite(B1,B2,User) :-
    get_value(B1,members,Ms), member(User,Ms), get_value(B1,systems,SYSSs),
    identify(D), member(D,SYSSs), get_value(B1,iswriting,Rs), length(Rs)>1,
    get_value(B1,leader,L), User=L, remove(X,iswriting,NL), append(User,NL,NL2),
    set_value(B1,iswriting,NL2,B2)),
  (can_endwrite(B1,B2,User) :-
    get_value(B1,history,H), today(D), append([User,endwrite,D],H,H2),
    set_value(B1,history,H2,B2), get_value(B1,iswriting,Rs), member(User,Rs)),
  (cangrant(B1,B2,User1,User2,Sys) :-
    get_value(B1,leader,L), User1=L, get_value(B1,members,Ms), append(Ms,User2),
    set_value(B1,members,Ms,B2), get_value(B1,systems,SYSSs), identify(D1),
```

```

member(D1,SYSSs), not(member(Sys,SYSSs)), append(Sys,SYSSs),
set_value(B1,systems,SYSSs,B2)) ],
[ (creator=fred), (leader=han), (members=[fred,han,greg]),
(systems=[univ_twente]), (isreading=[]),(iswriting=[]), (history=[])) ]

```

The binding `members` is a list of members who have rights on this document. The binding `systems` is a list of system environments that are permitted for the members to access the document. This binding prevents the document from being accessed from an untrusted environment. The license bindings `isreading` and `iswriting` are two sets that indicate the users who are reading and writing the document currently, respectively. In other words, they indicate the current status of the document. The license binding `history` functions as audit trail that records operations that have been performed by the users on the document.

There are 5 rules involved in this scenario, as can be seen as follows:

```

startread(Doc,User)           : license(Doc,C,B1) -> license(Doc,C,B2)
                               <= C |- can_startread(B1,B2,User)
endread(Doc,User)             : license(Doc,C,B1) -> license(Doc,C,B2)
                               <= C |- can_endread(B1,B2,User)
startwrite(Doc,User)          : license(Doc,C,B1) -> license(Doc,C,B2)
                               <= C |- can_startwrite(B1,B2,User)
endwrite(Doc,User)            : license(Doc,C,B1) -> license(Doc,C,B2)
                               <= C |- can_endwrite(B1,B2,User)
grant(Doc,User1,User2,System) : license(Doc,C,B1) -> license(Doc,C,B2)
                               <= C |- cangrant(B1,B2,User1,User2,System)

```

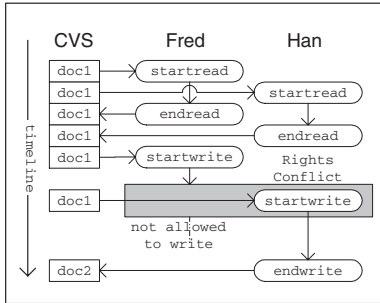


Fig. 3. A timing diagram of rights (real-time) concurrent activities in the document sharing system.

as far as we know. Additionally, XrML and ODRL do not provide status constraint to describe the real-time condition of the content.

This scenario exhibits a concurrent content access pattern: More than one subject may be performing the same operation on the same content simultaneously. This evokes conflicts.

For instance, Fig. 3 shows that when Fred and Han attempt to write at the same time, the project policy prevails, where it states that the project leader possesses higher privilege, Han can overwrite the document. XrML and ODRL do not provide mechanisms for concurrent access control,

5.2 Licenses Evolution Modelling

Jack is designing a new business which involves creating offers and licenses. He is interested in knowing how the distribution of the licenses between numerous entities would influence the licenses evolution.

He would like to check if users are allowed to assert desirable rights, e.g. a *print* right, at their systems, and he wants to know if conflicts, i.e. the asserted right conflicted with the existing right stated in the license, could arise that might undermine the business. For instance, a user asserts a new *view* right which allows her to view the ebook indefinitely.

The offer Jack constructs is:

```
offer(ebook:a_fiction,
[ (canview(B1,B2,User) :-
  get_value(B1,user,User1), User=User1, get_value(B1,viewed_times,PT),
  get_value(B1,max_views,Max), PT<Max, get_value(B1,expires,Exp),
  today(D), Exp>D),
  (cancopy(B1,B2,User) :-
  get_value(B1,user,User1), User=User1, get_value(B1,expires,Exp),
  today(D), Exp>D),
  (cangive(B1,B2,User1,User2) :-
  get_value(B1,user,User), User1=User, set_value(B1,user,User2,B2),
  set_value(B1,viewed_times,0,B2)),
  (canagree(B1,B2,User) :-
  today(D), Exp is D+366, set_value(B1,expires,Exp,B2),
  set_value(B1,user,User,B2)),
  (canassert(C1,C2,B1,B2,Clause,Binds,User) :-
  get_value(B1,user,User1), User=User1, append(C1,Clause,C2),
  append(B1,Binds,B2)) ],
[ (viewed_times=0), (max_views=10), (expires=20/12/2005), (user=anyone) ]]
```

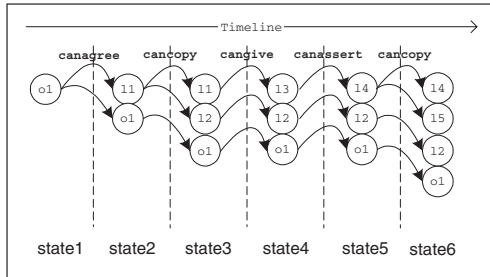


Fig. 4. A state chart of an example of license evolution in this scenario.

The license interpreter records each state of the license evolutions (i.e. each new license generated as a result of the evolution caused by the rules), including the newly generated licenses and the original licenses.

Additionally, through multiset rewriting the license interpreter is able to simulate the communications of more than two entities, for instance, content providers and content users. The license interpreter is able to help Jack tracing the logical design errors in his rights management system. Jack constructs the corresponding rules for the analysis on the evolutions of this license:

```
agree(Ebook,User)      : offer(Ebook,C,B1) ->
                        :   offer(Ebook,C,B1), license(Ebook,C,B2)
view(Ebook,User)       : <= C |- canagree(B1,B2,User)
                        :   license(Ebook,C,B1) -> license(Ebook,C,B2)
copy(Ebook,User)       : <= C |- canview(B1,B2,User)
                        :   license(Ebook,C,B1) ->
                        :     license(Ebook,C,B1),license(Ebook,C,B2)
give(Ebook,User1,User2) : <= C |- cancopy(B1,B2,User)
                        :   license(Ebook,C,B1) -> license(Ebook,C,B2)
assert(Ebook,Clause,Binds,User) : <= C |- cangive(B1,B2,User1,User2)
                        :   license(Ebook,C1,B1) -> license(Ebook,C2,B2)
                        :   <= C1 |- canassert(C1,C2,B1,B2,Clause,Binds,User)
```

The evolution of this offer may take the form of Fig. 4. The license 14 generated (at state 5 for user a who asserts a *print* right on the license):

The clause `canassert` of this offer determines if the user has the privilege to add right (i.e. clause, the argument `Clause` and the bindings, the argument `Binds`) to this license. The clause `canagree` transforms this offer to a license for the user, when the user agrees on the terms and conditions stated in this offer.

Jack can use his license interpreter (simulator) to analyze the li-

```

license(ebook:a_fiction,
[
  (canview(B1,B2,User) :-
    get_value(B1,user,User1), User=User1, get_value(B1,viewed_times,PT),
    get_value(B1,max_views,Max), PT<Max, get_value(B1,expires,Exp),
    today(D), Exp>D),
  (cancopy(B1,B2,User) :-
    get_value(B1,user,User1), User=User1, get_value(B1,expires,Exp),
    today(D), Exp>D),
  (cangive(B1,B2,User1,User2) :-
    get_value(B1,user,User), User=User1, set_value(B1,user,User2,B2),
    set_value(B1,viewed_times,0,B2)),
  (canagree(B1,B2,User) :-
    today(D), Exp is D+366, set_value(B1,expires,Exp,B2),
    set_value(B1,user,User,B2)),
  (canassert(C1,C2,B1,B2,Clause,Binds,User) :-
    get_value(B1,user,User1), User=User1, append(C1,Clause,C2),
    append(B1,Binds,B2)),
  (canprint(B1,B2,User) :-
    get_value(B1,user,User1), User=User1) ],
[ (viewed_times=0), (max_views=10), (expires=20/12/2006), (user=a) ])

```

The license interpreter is able to generate all possible licenses and offers. As can be seen at state 2, the *copy* right is performed on the old license (11) to generate a new license (12) in addition to the original license (11) at state 3.

The license interpreter can in principle be used as a licensing model checker, and the LicenseScript language *can be used as model language to specify the licensing processes*. In short, LicenseScript is a potential modelling language for licensing processes.

Additionally, LicenseScript *allows dynamic generation of new vocabulary for the rights expression in the license clause*, as shown in this scenario. XrML and ODRL do not support this feature. However, this has to be implemented with great care because this could be abusively exploited, which undermines the rights management system. We may control the rules with care, to render the rules trusted.

6 Conclusions and Future Work

We have presented an anatomy of right expression languages (RELs). In addition, we have studied the scenarios presented in XrML and ODRL, and we have translated them into LicenseScript (the REL we proposed in [3]). We have also studied novel scenarios and formalized them in LicenseScript.

This investigation is useful for understanding the strengths and weaknesses of ODRL, XrML and LicenseScript, and for assessing their capability of describing a number of important content access and distribution patterns as well as licensing processes. We have also demonstrated that LicenseScript is sufficiently flexible and expressive to capture the scenarios studied so far.

In LicenseScript one can define a new vocabulary for rights expression in the license clause, which XML-based RELs cannot support (as shown in scenario of license evolution modelling, section 5.2). We believe that this feature of LicenseScript may support copyrights enforcement in the rights management system. This deserves further study.

Acknowledgement. We like to thank Ernst-Jan Goedvolk from Telematica Instituut, Mark Stefik from Xerox PARC and our LicenseScript teammates for their valuable help. This work is sponsored by Telematica Instituut, The Netherlands.

References

1. J-P. Banâtre, P. Fradet, and D. L. Métayer. Gamma and the chemical reaction model: Fifteen years after. In C. Calude, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Multiset Processing (WMP)*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer-Verlag, Berlin, August 2001.
2. L. J. Camp. DRM: doesn't really mean digital copyright management. In *Proceedings of the 9th ACM conference on Computer and Communications Security*, pages 78–87. ACM Press, 2002. ISBN: 1-58113-612-9.
3. C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A novel digital rights language and its semantics. In *3rd International Conference on Web Delivering of Music (WEDELMUSIC)*, page to appear, Los Alamitos, California, United States, September 2003. IEEE Computer Society Press.
4. C. N. Chong, S. Etalle, and P. H. Hartel. Comparing logic-based and XML-based Rights Expression Languages. Technical Report TR-CTIT-03-30, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, July 2003. <http://www.ctit.utwente.nl/publications/Tr03/>.
5. R. Corin, C. N. Chong, S. Etalle, and P. H. Hartel. How to pay in LicenseScript. Technical Report to appear, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, July 2003. ISSN 1381-3625.
6. C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, pages 4034–4038, Maui, Hawaii, United States, January 2001. IEEE Computer Society Press.
7. H. Guo. Digital rights management (DRM) using XrML. In *T-110.501 Seminar on Network Security 2001*, page Poster paper 4, 2001. ISBN 951-22-5807-2.
8. R. Iannella. Open digital rights management. In *World Wide Web Consortium (W3C) DRM Workshop*, page Position paper 23, January 2001.
9. D. Mulligan and A. Burstein. Implementing copyright limitations in rights expression languages. In J. Feigenbaum, editor, *Proceedings of 2002 ACM CCS-9 Workshop on Security and Privacy in Digital Rights Management*, volume 2696 of *Lecture Notes in Computer Science*, page To appear. Springer-Verlag, November 2002. ISBN 3-540-40410-4.
10. D. Parrott. Requirements for a rights data dictionary and rights expression language. Technical Report version 1.0, Reuters Ltd., 85 Fleet St., London EC4P 4AJ, June 2001. In response to ISO/IEC JTC1/SC29/WG11 N4044: “Reissue of the Call for Requirements for a Rights Data Dictionary and a Rights Expression Language” – MPEG-21.
11. R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *IEEE Proceedings of the Computer Security Foundations Workshop*, pages 282–294, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.
12. B. Rosenblatt, B. Trippe, and S. Mooney. *Digital Rights Management: Business and Technology*. John Wiley & Sons, New York, United States, November 2002. ISBN 0764548891.
13. P. Samuelson. Digital rights management {and,or,vs.} the law. *Communications of ACM*, 46(4): 41–45, April 2003.

Simplifying CORBA Security Service to Support Service Level Access Control

Atul Kumar, Pankaj Jalote, and Deepak Gupta

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
208016, India
atul@iitk.ac.in

Abstract. The CORBA Security Service provides a domain based access control model in which interfaces are grouped in domains and a single security policy applies to all the interfaces in the domain. CORBA Security Service does not directly support object and method level access control rules. Grouping interfaces in domains helps in reducing access control rules in a large system with a large number of objects. In an environment where services are provided using methods and similar services are grouped as the methods of an interface, providing service and interface level access control is desirable.

We present an access model that works on the top of the access model of CORBA Security Service and allows method and object level access control for CORBA objects. A GUI tool has been developed for managing the users, defining user groups (*e.g.*, grouping users playing a certain role), and specifying the access control rules. The GUI tool is supported by a software layer which we have built on top of CORBA Security Service. Our access model coexists with the CORBA Security Service access model and the complex features of CORBA Security Service can be used wherever required.

1 Introduction

Security and business reasons require access control for certain services in a network based applications. In a distributed object system, services are provided using objects. A single object can provide more than one service as its interface can have more than one method. This may require different access control policies for the different methods of the same object. At the same time, many operations which may be implemented as a number of methods in several objects may be the parts of the same service and therefore require a common access control policy.

CORBA Object Services specifications include a Security Service specification. The CORBA Security Service [13] provides a domain based security model in which interfaces are grouped in domains and a single security policy applies to all the interfaces in the domain. Methods of an interface are listed under one or more of the four rights defined in the default *RightsFamily* called *corba*, which

has four rights labeled as **g** (get), **s**, (set), **u** (use) and **m** (manage) [4]. Grouping interfaces in domains help in reducing access control rules in a large system with a large number of objects. In an environment where services are provided using methods and interfaces, method and interface level access control is desirable. Moreover, in a small setup, where access to certain services is granted to individual users (or to the group of users), security administrators may find it difficult to understand (and manage) the CORBA Security Service terminologies such as *domains*, *rights*, *principals*, *privileges*, *roles*, *protection*, and *delegation* etc.

In this paper, we present an access model that allows method and interface level access control for CORBA objects. A GUI tool has been developed for managing the users, defining user groups (*e.g.*, grouping users playing a certain role), and specifying the access control rules. The tool is supported by a software layer which we have built on top of CORBA Security Service. Our access model coexists with the CORBA Security Service access model and the complex features of CORBA Security Service can also be used wherever required.

Before discussing our access control model, we first briefly discuss various terminologies used in the area of information security. We also discuss the access control model of CORBA Security Service.

2 Security and Protection

In early days of computing, hardware and software resources and the people having access to those resources were limited [1, p. 1]. So computer security was of little concern. Computer security related concerns emerged when computers began to be used for storing and processing classified information. Confidentiality was the primary security concern [14, p. 9]. As computers became more available and the sharing of resources became possible through the networks, additional security problems surfaced. Operating Systems were required that could prevent users from interfering with each other, intentionally, accidentally or ignorantly [11, p. 15]. Computer security can be defined as “*keeping anyone from doing things you do not want them to do to, with, on, or from your computers or any peripheral devices*” [6, p. 3]. Security techniques can be defined as the “*techniques to help keep your system safe from other people - including both insiders and outsiders, those bent on destruction, and those who are simply ignorant or untrained*” [11, p. 6]. A taxonomy for the classification of computer and network security attacks and incidents is given in [12].

Protection is usually provided by a set of mechanisms. Each mechanism or countermeasure is designed to prevent a specific kind of unwanted event from happening. For example, a file system access control mechanism is designed to prevent users from reading and writing files they are not supposed to read or write. This approach to protection assumes that it is possible to list most of the unwanted events which can happen in a system, and it is possible to build mechanisms to prevent those things from happening [4, p. 4].

3 Security Policy and Security Models

A security policy is a formal statement of the rules by which people who are given access to an organization's technology and information assets must abide [10]. It must be implementable through system administration procedures and it must be enforceable with security tools.

Generally, a security policy is expressed in words and emulates real world security challenges as a set of laws, rules, roles and procedures defining how an organization allocates resources to protect assets and achieve business security objectives [15, p. 103].

Security policies normally are of three types, each with a different purpose: organizational, issue, and system specific [5]. An organization's security policy is the management's security strategy for protecting assets and attaining an enterprise's business security objectives. It outlines specific employee roles, responsibilities, rules and procedures for compliance. An issue or technology specific policy outlines similar responsibilities, details and procedures dealing with a specific technology or system like e-mail, physical security, software, Internet, or LAN. Lastly, a system specific policy must address internal computer system or an application's technical mechanisms (*e.g.*, automated system or application security policy) that control the security access policy for subjects and objects within a specific computer system. Security policies vary from organization, business, environments, technical application and network systems.

A computer security models restate the security policy in more formal or mathematical terms to: (1) provide framework for understanding concepts; (2) provide an unambiguous, formal representation of a general security policy; (3) express the policy enforced by a specific computing system [15, p. 110].

Security models may be formal or informal. Formal models are based upon logic and formal mathematics, information or complexity theory. A common formal model known as Bell LaPadula model [2], features latticed information flow, strict labeling and mandatory access control emphasizing confidentiality for secrecy. In commercial and medical systems, Biba [3] and Clark - Wilson [7] integrity oriented models are common and focus on integrity levels of information flow for transaction accuracy and separation of duty. Informal models can be precise narratives of system operations supplemented with concept facilitating diagrams, graphs or tables like an access matrix. Whether formal or informal, models are a means to abstract essentials from the typically ambiguous policy narrative to precisely represent and better understand the policy and evaluate assurance of the system's security.

Computer *security policy* is a set of requirements for a specific system; a *security model* is a restricted representation of a class or types of systems that abstracts unneeded details to highlight a specific property or set of behaviors. Models are useful to design and better understand a specific system policy and provide a general framework for understanding a type of system's implementation of security mechanisms employed to enforce a defined policy [1, p. 92].

4 CORBA Security Service

CORBA distributed object system, like other systems, addresses the security requirements by applying protection based on some policy. The CORBA security model [13] defines a framework for security in distributed ORB-based systems. It defines a framework in which multiple implementation of security policies may reside. Since security is a term used to aggregate a large set of features, the possible security and security methodologies are quite numerous. It is not possible to assume that one single solution will fit all. Therefore, the CORBA security model serves as a framework that defines several security levels and an implementation can choose to conform to the specifications at any of the levels.

4.1 Authentication

A *principal* is a human user or system entity that is registered in and authenticated to the system. Initiating principals are the ones which initiate activities. An initiating principal may be authenticated in a number of ways, the most common of which for human users is a password. For system entities, the authentication information such as its long term key, needs to be associated with the object.

An initiating principal has at least one, and possibly several identities (represented by the system using attributes, *i.e.*, audit identity, access identity). The principal may also have privilege attributes which can be used to decide what it can access. At any one time, the principal may be using only a subset of these permitted attributes, either chosen by the principal (or an application running on its behalf), or by using a default set specified for the principal. There may be limits on the duration for which these privilege attributes are valid and may be controls on where and when they can be used. Security attributes may be acquired in three ways:

1. without authentication (Public);
2. through authentication (identity attributes and privilege attributes);
3. through delegation from other principals.

When a principal is authenticated, it normally supplies its security name, authentication information, and the requested privilege attributes. A principal must establish its credentials before it can invoke a secure object. If the principal has no credentials at the time of the request and the service requires credentials, the client must invoke the *Principal Authenticator* object to authenticate, and optionally select attributes for the principal for this session. This creates the required *Credentials* object and makes it available as the default *Credentials* for this client.

4.2 Access Control

The CORBA security model allows a range of access policies and control attributes which can group subjects (using privileges), objects (using domains)

and operations (using rights). The specification defines a particular access policy type and associated management interface as part of the security functionality level

Regardless of the access control policy management interface used, all access decisions on object invocation are made via a standard access decision interface. However, different management interfaces will ordinarily be required for management of different types of control attributes. The standard access decision interface belongs to the *AccessDecision* object. It is used at runtime to perform access control checks. *AccessDecision* objects rely upon *AccessPolicy* objects to provide the policy information upon which their decisions are based.

The standard *AccessPolicy* objects in a secure CORBA system implements access policy using rights. In rights-based systems, *AccessPolicy* objects grant rights to *PrivilegeAttributes*; for each operation in the interface of a secure object, some set of rights is required. Callers must be granted these required rights in order to be allowed to invoke the operation. To allow extensibility, rights are grouped into rights families. The *RightsFamily* containing the standard rights is called *corba* and contains four rights: *g* (get), *s* (set), *u* (use), and *m* (manage). Implementations may define additional right families. Rights are always qualified by the *RightsFamily* to which they belong.

Domains are groups of objects. A domain contains one or more objects plus a set of access control rules called policy. Each *Domain* is assigned an *AccessPolicy*, which is applied to all objects in the domain. Each access-controlled object in a CORBA system must be a member of at least one *Domain*. A *DomainAccessPolicy* object defines the access policy which grants a set of named *subjects* (e.g., users) a specified set of *rights* (e.g., g,s,u,m) to perform operations on the objects in the domain.

5 A Model for Method and Object Level Access Control

The CORBA Security access control model, due to its approach of grouping objects (into domains) and methods (into rights), makes managing of access control rules easy in a system with a large number of objects with many subjects accessing their methods. However, in certain situations, a fine grained access control mechanism may be required to control access to an individual service.

In our model we consider an object as the provider of a set of services, each service being implemented by a method. To provide service-level access control, we need to support a method level security model. As an object is likely to provide many related services, which an object may want to provide together, we also want to support an object-level security model. With both an object-level and method level security models, access can be given either to a complete object or it can be given on individual method of an object.

The proposed model has following constituents:

- **Subjects:** A subject in this model is a user similar to the user of CORBA Security Service. A subject is identified by its unique identifier (e.g., user

name). One or more subjects can together form a group of subjects. Grouping can be on the basis of a common role, or it can be just an arbitrary collection of subjects. Groups are also assigned unique names to identify them in the access control rules.

- **Objects:** An object in this model is simply a CORBA interface.
- **Operations:** An operation is just a method of a CORBA interface.

With these components, our goal is to support a security model such that a user or a group of users can be given some specific rights to a method of an interface or to an entire interface (that is, access to all methods of the interface). A fine grained security policy with rules for method and interface level access can be implemented using this model

If a subject (user or user group) is granted access to an interface, it means that the subject can invoke all the methods on an object of that interface. If a subject is granted access to a specific method of an interface, it means that the subject can execute that method on an object of the interface. The subject may not be able to invoke other methods of the interface if it has not been given the explicit permissions on the other methods of that interface.

With this, we have many types of access control rules. An access control rule tells the system which subject is given access to which operations on object. The following are the possible types of access control rules in this system.

user u is granted access to interface i
group g is granted access to interface i
user u is granted access to method m of interface i
group g is granted access to method m of interface i

The above rules can coexist with the access control rules of CORBA Security Service which supports the rules of the following form:

user u is granted rights s, m on domain D
group g is granted rights g, s on domain D

This is a simple framework for providing security which gives a fine grained control to the service providers. In this model, it is the service provider that specifies which of the services it is providing are accessible and to whom. Presumably the service provider will provide the access based on some commercial considerations like charging for services. Though the commercial or payment aspects are outside our scope, this model provides the necessary mechanisms for it.

The challenge now is to implement this security model on top of the CORBA's Security model. That is, we do not wish to change the security model provided by CORBA (and its implementation). Rather, we would like to translate the access rules in our model into CORBA security model and then use CORBA Security Service to support our security model.

The default *RightsFamily* of CORBA Security Service called *corba* has four *required rights*: **g** (get), **s** (set), **u** (use), and **m** (manage). Each of these has an intended meaning. **g** is supposed to group methods which return information to the caller; **s** is supposed to group methods which change information inside an object; **u** is supposed to be used to group methods which cause an object to do some kind of work; and **m** is supposed to be used to group methods which normal users should not be calling, there may be methods intended for use by system administrators. The CORBA Security Service allows administrators to define additional right families with more rights. Our model takes advantage of this feature to define as many new rights as required for categorizing the different services provided by an object.

6 Implementation

We have implemented the proposed access model in Java using JDK 1.3 of Sun Microsystems on Linux and Solaris boxes as the development environment. We use the CORBA class libraries of JavaORB, an open-source CORBA 2.3 implementation [9]. JavaORB provides an Object Security Service (OSS) [8] which is an open-source implementation of the CORBA Security Service specification.

A security policy implementation should permit objects to remain unaffected, if they do not wish to have security. On the other hand, the objects that require security and access control, should be provided that. In other words, the implementation should permit insecure object as well as secure objects to coexist. The insecure objects and their access should be same as it would be without a security service. The overall architecture of the system with our security service is shown in Figure 1. The blue boxes are the components implemented by us while the white boxes represent existing components.

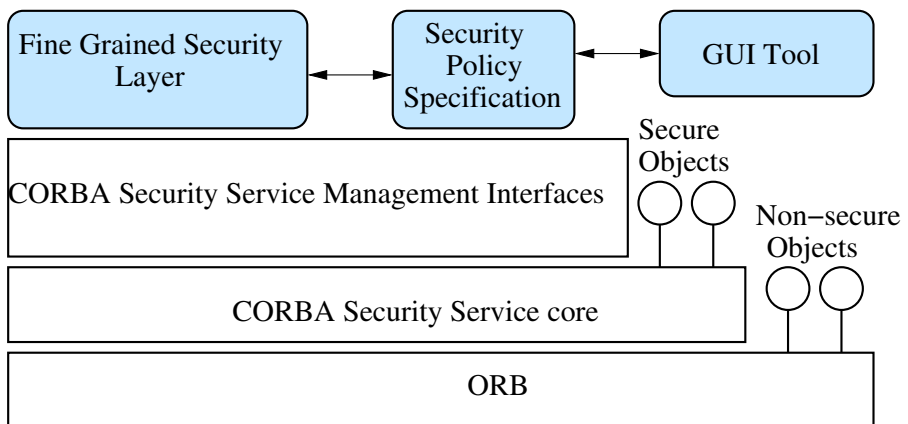


Fig. 1. Fine Grained Security Model

As can be seen, our implementation consists of two parts: specification of the security policy (access control rules) and then translating the specified access control rules into CORBA Security Service.

The fine grained security layer reads the previously defined specifications from the configuration files and builds a CORBA Security Service equivalent model from this. It then queries the CORBA Security Service Management interfaces to check if the whole or a part of this implementation is already registered with the CORBA Security Service. The registration might have been done by the previous invocation of this module. Appropriate requests are then sent to the CORBA Security Service to support the specifications.

The security policy specification component is a GUI tool that has been developed for allowing the security administrator to easily specify the access control policy. We have used the Swing GUI components of Java Foundation Classes (JFC) to build the GUI tool.

The GUI tool writes user specified access control policy and its CORBA Security Service translation to files on the disk. Saving the user specified access policy and its mapping to CORBA Security Service is necessary for allowing the security administrators to edit the previously specified policies. Since CORBA Security Service does not directly support object and method level access control rules, it is not possible to retrieve the fine grained access control rules specified using the GUI tool just by querying the CORBA Security Service. The Security administrator does not need to know about the mapping of user specified access policy to CORBA security service and the file in which it is saved. The tool automatically takes care of access rule translation and file name management. An administrator can add new access control rules to the list of previously specified rules dynamically. Similarly, rules can be modified or deleted dynamically. The tool makes changes in the corresponding files immediately when some rules are added, deleted, or modified and the fine grained security layer sends corresponding requests to the CORBA Security Service management interfaces so that these changes get reflected in the CORBA Security Service.

Fine grained access control rules are translated to the CORBA Security Service access control rules as follows.

- A user of our model is mapped to a user of CORBA Security Service directly (with the same user name).
- A user group is mapped to a *Privilege* of type *group* and this *Privilege* is added to all the members of that user group.
- An access control rule of the form “user *x* is granted access to interface *i*” is translated as follows. If this is the first access control rule granting access to interface *i*, then a new domain is created with *i* as its domain name. If the domain name *i* is already in use then the first available name from *i1*, *i2*, ... is used. Interface *i* is added as the only member in this domain. All the methods in interface *i* are listed under the right *u* (of default *RightsFamily*). If at least one access control rule granting access to interface *i* has been encountered previously, then the domain that was created to translate that rule is used instead of creating another domain. Suppose that the newly

created domain (or the previously created domain) for interface **i** has been given a domain name **i**. Finally the following rule is added to the access control policy of domain **i**: “user **x** is granted right **u**”.

- An access control rule of the form “group **g** is granted access to interface **i**” is translated as follows. A domain is created or a previously created domain is identified for interface **i** in the same manner as discussed above. Suppose that the domain name is **i**. Now the following rule is added to the access control policy of domain **i**: “privilege **g** is granted right **u** on domain **i**”, here **g** is the *Privilege* of type *group* that has been created to map group **g** into CORBA Security Service.
- An access control rule of the form “user **x** is granted access to method **m** of interface **i**” is translated as follows. A domain is created or a previously created domain is identified for interface **i** in the same manner as described above. If this is the first method-level access control rule for the interface **i**, then a new *RightsFamily* **i** is created with the same number of rights as the number of methods in the interface **i**. These rights are labelled as **a**, **b**, **c**, ... (only one character is allowed as a label of a right in a *RightsFamily*). If at least one method-level access control rule granting access to interface **i** has been encountered previously, then the *RightsFamily* that was created to translate that rule is used instead of creating another *RightsFamily*. Suppose that the right associated with method **m** is **a**. The following rule is added to the access control policy of domain **i**: “user **x** is granted right **a**”.
- Access control rules granting method-level access to user groups are translated similarly with user replaced by the *Privilege* (that was created for mapping the user group specified in access control rule) in the access control policy of the domain.

The tool maintains the fine grained access control information and its mapping to CORBA Security Service in a data structure that is stored in various files on the disk so that the previously saved configuration can be read when the tool is started. CORBA Security Service management interfaces are queried to verify the consistency of the configuration read from the files. Interface definitions are obtained by querying the *Interface Repository*. When some changes in

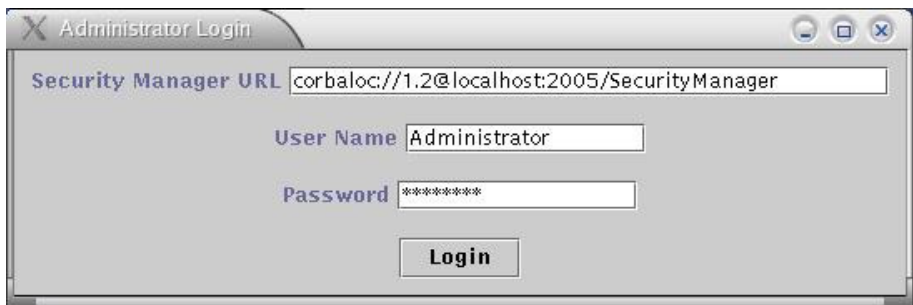


Fig. 2. Administrator Login Screen

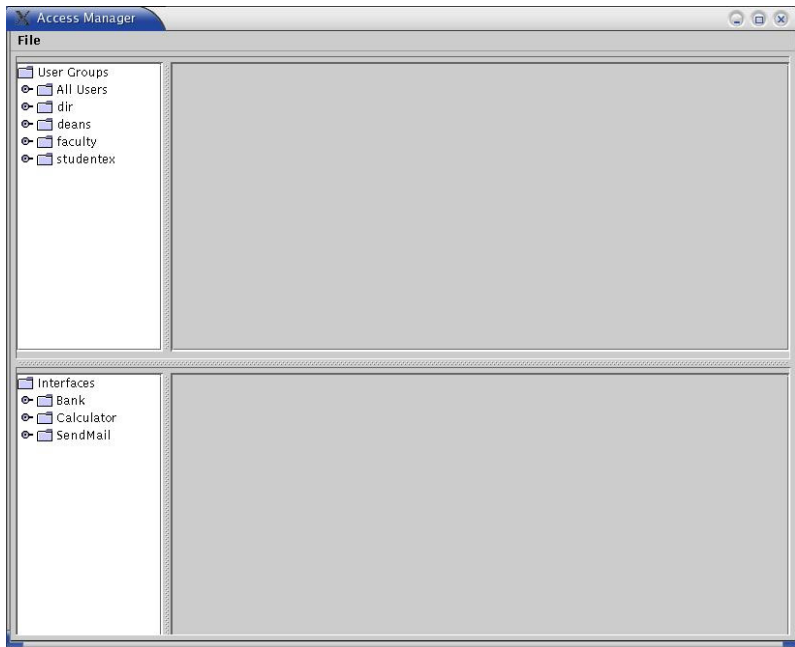


Fig. 3. Startup View of GUI Tool

the security system are made by the security administrator using this tool, these changes are translated to CORBA Security Service using the mapping discussed above, and updates are made to the CORBA Security Service database using the management interfaces provided by the CORBA Security Service. This requires the administrator password of the *SecurityManager*. The GUI tool provides a screen for administrator login (Figure 2).

The main window of the GUI tool is split horizontally (Figure 3); the top half is used to manage users and user groups, and bottom half is used to manage access control of interfaces and their methods. The GUI tool has following features.

- It allows defining new user groups and editing/deleting existing user groups (Figure 4).
- It allows adding new users and editing/deleting existing users (Figure 5).
- It allows users and user groups to be added to the access list of an interface (Figure 6).
- It allows adding interfaces to a selected user or user group. Adding an interface to a user or user group means granting access to all the methods of that interface (Figure 7).
- It allows method level access control of an interface to a user or user group (Figure 8).
- It allows users and user groups to be added to the access list of a method in an interface (Figure 9).

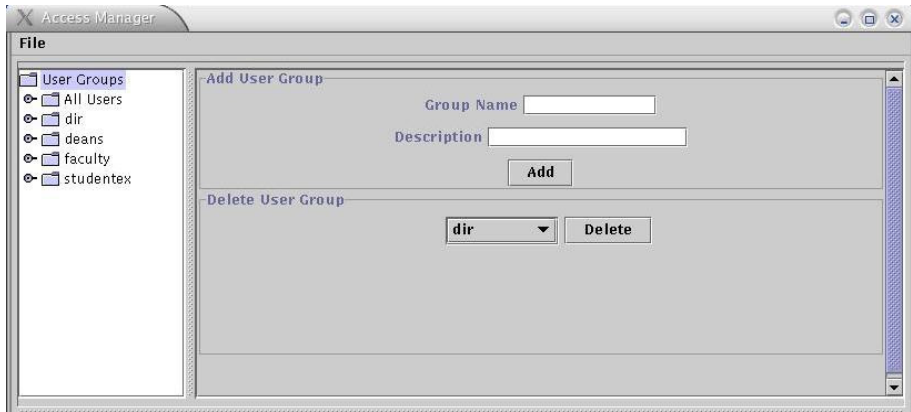


Fig. 4. Managing User Groups

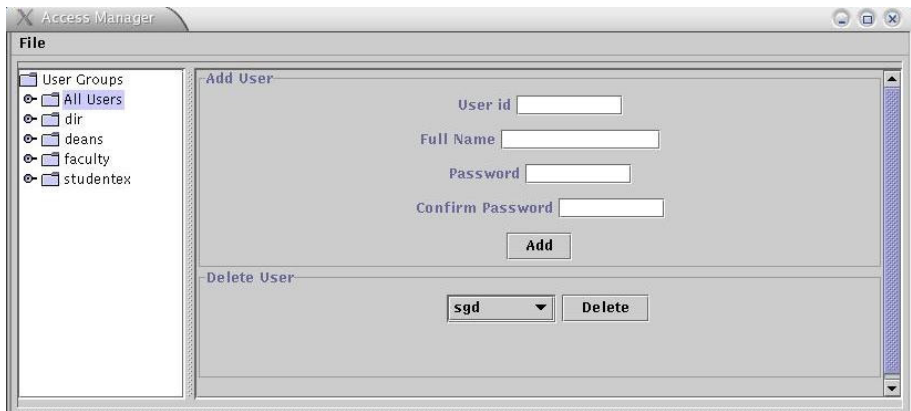


Fig. 5. Managing Users



Fig. 6. Managing Access List of an Interface

The proposed security model and the GUI tool that implements it present a simple view of CORBA security to the security administrators. For simple applications, where users or groups of users are given access to individual services (methods) or the set of related services (methods in an interface), the security administrators can grant these accesses directly. This hides the complexity of CORBA Security service where the security administrator has to manage domains, principals (which can be users or services or something else), privileges, *RightsFamilies* and rights, and finally define and manage the access control policy for each domain. Since the proposed security model coexists with the CORBA security model, the advanced features provided by CORBA Security Service can be used where the access policy requirements are complex and cannot be handled by the proposed model.

7 Conclusion

A security policy is a description of the needs and requirements of a system to protect against various threats to the system. This is a high level declarative description reflecting the security problems faced by the system. A security model is a representation of a security policy and deals with the security properties (such as rules and constraints) of the systems. The security model is abstract and generic and should only contain information pertinent to the security aspects of the system.

The CORBA Security Service provides domain based access control mechanism to reduce the number of rules in the access control policy in a large system having many objects. This mechanism does not directly provide object and method level access control.

In some situations, it may be desirable to have a finer level of granularity. We have proposed a finer level of access control for CORBA environment which allows a security administrator to specify object and method level access control

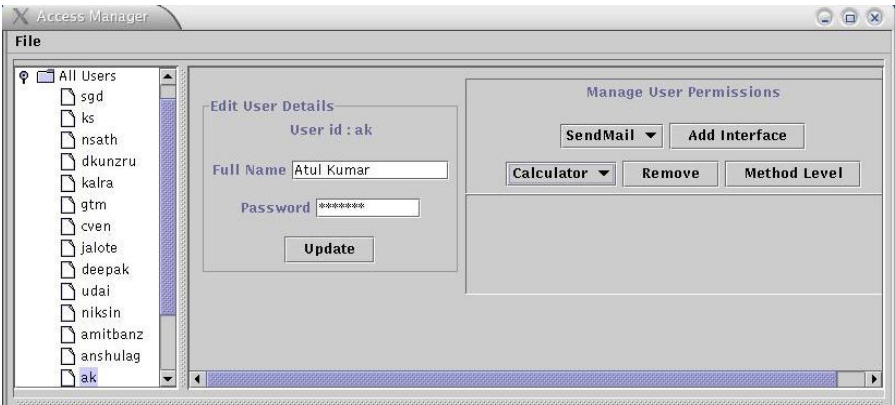


Fig. 7. Managing Individual User's Access

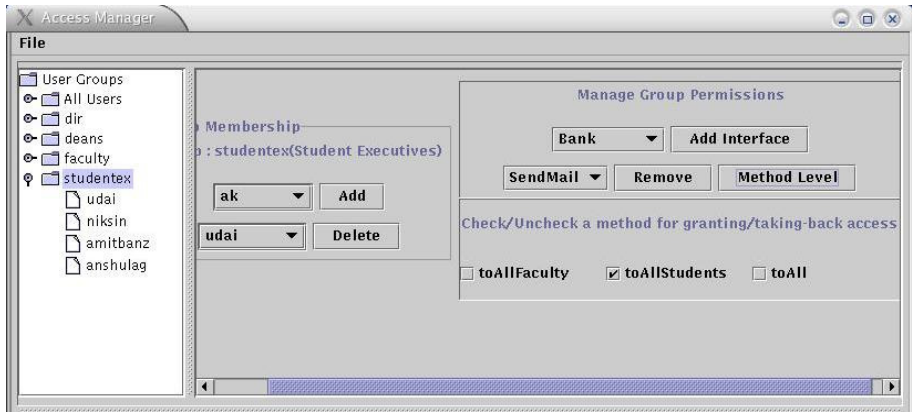


Fig. 8. Method Level Access for an Interface

rules in the security policy. This is implemented by creating domains with a single interface as its member and then defining additional right families to have one right each for each of its methods. We have developed a GUI tool which transparently manages the creation and assignment of domains and right families. With our tool, application developers and security administrators are not required to understand and work with the complex CORBA Security Service interfaces.

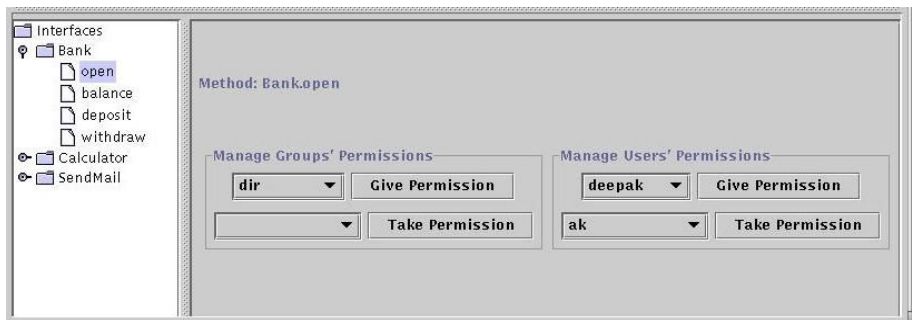


Fig. 9. Managing Access List of an Individual Method

References

1. Edward G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1994.
2. D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Technical report, MITRE Corporation, Bedford, MA, USA, 1973.
3. K. Biba. Integrity considerations for secure computer systems. Technical report, U.S. Air Force Electronic Systems Division, 760372, 1977.
4. Bob Blakley. *CORBA Security An Introduction to Safe Computing with Objects*. Addison-Wesley, Reading, MA, USA, 2000.
5. Murkier Brew. Java's evolving security model: Beyond the sandbox for better assurance or a murkier brew? SANS Institute Technical Report, March 2001.
http://www.giac.org/practical/gsec/Matthew.Herholtz_GSEC.pdf.
6. William R Cheswick and Steven M. Bellolovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Publishing Company, 1996.
7. D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, USA, May 1987.
8. Distributed Object Group (DOG). JavaORB Object Security Service (OSS).
http://dog.team.free.fr/details_oss_service.html.
9. Distributed Object Group (DOG). JavaORB version 2.2.7.
http://dog.team.free.fr/details_javaorb.html.
10. B. Fraser (editor). Site security handbook. RFC 2196, IETF, September 1997.
<http://www.ietf.org/rfc/rfc1945.txt>.
11. Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet Security, second edition*. O'Reilly & Associates, 1996.
12. John D. Howard. *An Analysis Of Security Incidents On The Internet 1989 – 1995*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, April 1997.
<http://www.cert.org/research/JHThesis/Start.html>.
13. Object Management Group. Security service specification, version 1.7. OMG Document formal/01-03-08, March 2001.
<http://www.omg.org/docs/formal/01-03-08.pdf>.
14. Deborah Russel and G. T. Gangemi, Sr. *Computer Security Basics*. O'Reilly & Associates, 1991.
15. Rita C. Summers. *Secure Computing: Threats and Safeguards*. McGraw Hill, New York, NY, USA, 1997.

The Role of Deontic Concepts and Commitment in the Policy Specification of Mobile Computing Environment

Xiaofeng Gong and Julian Newman

Division of Computing
School of Computing and Mathematical Sciences
Glasgow Caledonian University
70 Cowcaddens Road, Glasgow, G4 0BA, UK
{x.f.gong, j.newman}@gcal.ac.uk

Abstract. The existing authorization policy language and technologies in traditional distributed systems can not meet the requirements of dynamic ubiquitous computing environment. The provision of an efficient and flexible policy specification approach is needed to support the deployment of access control of mobile computing applications. This paper introduces a deontic concepts based approach for security policy specification in the wireless, mobile computing environment, considers in particular the problems that can arise when collaborative commitment are negotiated via agents communication, and give the description of relationship between obligation and commitment in the specific organization domains.

Keywords: Commitment, deontic concepts, commitments, speech acts oriented communication, security policy specification, mobile computing.

1 Introduction

Security policy is a formal statement of rules which regulate and distribute sensitive information. The research for modeling authorization policy has been discussed widely in the literature and has given rise to many solution models such as the discretionary access control model(DAC)[NCSC 1987], the mandatory access control model(MAC) [Bell et al 1975] and the role based access control model(RBAC)[Sandhu et al 1996]. Many issues have been discussed for using the logic based language for specifying the specific part in RBAC model. [Chen & Sandhu 1995] introduced a language based on set theory for specify the state-related constraints in RBAC, which can be later translated to a first-order predicate logic language. Another logic language - Role based Constraints Language (RCL2000) is introduced in [Ahn & Sandhu 2000] which is based on a restricted form of first order predicate logic is defined for encompasses obligation constraints in addition to the usual separation of duty and prohibition constraints. The language covers both static and dynamic separation of duty constraints.

The Cambridge OASIS architecture [Bacon, Moody & Yao, 2002][Hayton, Bacon & Moody 1998] describes an RBAC model which covers both aspects of RBAC theoretical modelling and practical deployment. It supports access control in heterogeneous distributed systems by managing roles through certificates. Cambridge

OASIS supports a horn clause based Role Definition Language for role modeling (in current implementations this is realized as XML files). Role based access control policies are presented by specifying three aspects of role membership: role declaration, role entry and role revocation. The main functions are role declaration for naming a role, role entry for activating a role and role revocation for revoking a role. Unlike Sanhdu's RBAC model, role activation in Oasis is based on the prerequisite role, not role hierarchy.

Those security models clearly and concisely specify the access rules for information systems. They provide the fundamental theoretical base for the deployment mechanisms of the models and there are many deployment technologies which have been developed for the based on the access control models. Those schemes well support authorization in traditional distributed systems, in which the user who makes the accessing requests can be predetermined.

However modern mobile computing infrastructures have turned the internet into a dynamic communication environment. It raised the more crucial security concern about accessing sensitive information over wireless networks. Moreover, in a mobile world individuals are intermittently connected, and a transaction or conversation at a business level may be conducted not only where a user is moving between domains, but may even involve a number of different principals or agents representing the same user at different times during the conversation or workflow. An enterprise-wide policy based security system requires a logically precise approach to define the authorization policy. This requirement encouraged researchers to search for a more expressive and unambiguous language to reason about access control policy. In this paper, we will introduce elements of a policy specification language based on dynamic deontic logic. We will introduce a deontic-logic based approach for security policy specification in the wireless environment, particularly focussing on the problems that can arise when collaborative commitments are negotiated via agent communication, and give a description of the relationship between obligation and commitment in the specific organizational domains. As Sergot [2001] notes, the term commitment has been used in multi-agent systems to refer to a wide range of concepts, but one common usage, which Sergot adopts and which we adopt here, is to refer to a directed obligation of one agent towards another.

2 Deontic Concepts

Deontic logic is a branch of modal logic, dealing with deontic concepts such as Obligation(*O*), Permission(*P*) and Forbidden(*F*). Ortolano [1998] discusses the use of deontic logic to specify access control policy based on deontic concepts. The theorems of standard deontic logic system (KD system), which is specified by the axioms presented in [Meyer 1988] and [Cholvy & Cuppens 1997].

The theorems of deontic logic include all (or enough) tautologies of Propositional calculus and the axioms and the KD system provides a means of analysing and identifying ambiguities in sets of legal rules. However, it is increasingly recognised that these are not purely legal concepts but relate to all kinds of regulated interactions amongst principals or agents (cf Sergot, 2001).

The logic language based on deontic concepts is also used to represent authorization policies with the aim of dealing with inconsistencies of authorization policies

in [Cholvy & Cuppens 1997] by translating the deontic logic representation into first order predicate logic before performing the necessary conflict analysis. In their previous work, they also describe show delegation can be represented using deontic notation. The approach is limited to expressing policies for agents in terms of what they are obliged, permitted, and forbidden from doing. However there are still a few inherent problems with the standard deontic logic system which are considered not suitable for the practical enterprise modelling of the logic and leads us to consider some new developments in the fields of deontic logic, agency and normative systems.

One of the problems is the existence of a number of paradoxes. A solution to eliminating the Paradoxes, has been proposed by Meyer [1988] who introduces a reduction of deontic logic to action logic. The reduction defines V as the violation atom, meaning a liability to some sanction or punishment as the result of an action. With the V atom, the deontic logic operations are defined as presented in the following,

Definition1: $F\alpha =_{\text{Def}} [\alpha]V$

Action α is forbidden if the performance of α yields a states where V holds

Definition2: $P\alpha =_{\text{Def}} \langle \alpha \rangle \neg V (= \neg F\alpha)$

Action α is permitted if action α is not forbidden (if there is some way to perform a that leads to a states where V does not hold)

Definition3: $O\alpha =_{\text{Def}} [-\alpha]V (= F(-\alpha))$

Action α is obligatory if not-doing α is forbidden

In the definition, the α represent a generic action, $-\alpha$ is the non-performance of α , $[\alpha]$ is the execution of α and $\langle \alpha \rangle$ is possible execution of α . The approach of reducing deontic logic to dynamic logic removes most of the paradoxes that characterising classical deontic logic [Meyer 1988]. Additionally, the approach to deontic logic facilitates their use in specifying integrity constraints for knowledge bases [Wieringa et al 1991]. This provides a sound basis for the practical use of the logic within a prototype system.

In our related work, a process model based on action logic, the reduced deontic logic, has been successfully formalized to support process co-ordination in virtual corporation [Haag, 2000; Haag, Foley & Newman, 1998]. An extended structure for integrating the commitment and deontic logic is presented as the integrating rules based on the two forms of abstraction of commitment in co-ordination support system and presented a deontic logic based formalism for capturing and managing commitment in the purpose of supporting co-ordination. See figure 1.

deontic_operator(actor, action, entity)

deontic_operator = $\{O \mid P \mid F\}$

entity = $\{\text{actor} \mid \text{target\_object}\}$

Fig. 1.

The work shows us a possible approach to formalize certain aspects of commitments with deontic logic concepts for policy rule specification. Note however that in the above work obligation remains generic rather than arising from specific negotiations or being an obligation towards an individual or collective actor. Thus Haag (op.cit) is able to model the mismatch between processes working across organizational boundaries in a virtual software corporation, but does not address the specific origin of obligations in speech acts within a conversation for action [Winograd and Flores, 1987]. In the next sections, we shall sketch the syntax of a deontic concept based policy structure and the description of relationship between obligatory and commitment based speech acts in the organization domains.

It is believed that deontic concept based policy formalization promises a more flexible and expressive way for policy specification. The proposed approach is to use deontic logic for policies was already discussed in several literatures and a valued review articles about deontic logic is in [Wieringa and Meyer 1998].

In our policy language, we defined a domain-dependent actions object for constructing the deontic concepts policy prohibitions, obligations, and forbiddens. The ActObjects are constructed as two parts (a, t), which a is a action-list for a target objects which are associated with certain domains.

2.1 Obligation

Obligation policies specified the responsibilities of the certain subjects have to be done when certain events occurs, for example, some actions have to be taken when certain violation occurs. It also regards as event triggered policies. The following policy states the event triggered obligation policies.

$\text{Obligation}(z, x, \text{ActObject}, \text{eventsCondition}, \text{constraints}) \vdash \neg (z, x, \text{ActObject}, \text{constraints}) \rightarrow V$

The policy expresses that a subject entities z , is obliged to execute the actions to x in ActObject when the event conditions are satisfied if not execute the actions, a violation holds. V means violation with the value: true or false.

2.2 Permission

Permission refers to some actions which are not obliged to execute by the correspondence subject. It specifies the possession of the rights which the entity holds associated with target objects in the certain domain.

$\text{Permission}(x, \text{ActObject}, \text{constraints}) \vdash (x, \text{ActObject}, \text{constraints}) \rightarrow \neg V$

The policy expresses that a subject entities x , can possess the rights to execute the actions iff x execute the actions, no violation V holds.

2.3 Forbidden

Forbidden states negative access rights, which is prohibited from performing action. It's more like the negative authorization policy which in Ponder Langue [imperial college)

$\text{Forbidden}(x, \text{ActObject}, \text{constraints}) \vdash (x, \text{ActObject}, \text{constraints}) \rightarrow V$

The policy expresses that a subject entities x , is forbidden to possess the rights to execute the actions iff x execute the actions, violation V holds.

In the policy specification, we can also accept the subject as a means of role based objects and domain dependent group objects to which policies apply.

3 Commitment and Speech Acts

It has been considered that many deontic logic based systems play the static feature towards the concept Obligation as they often arise from general ethical principles or standing laws which is more focus on the impersonal statement. However based on the social notion of commitment, we can describe a logic relationship between Obligation and Commitment. Colombetti [2000] considered commitment as a deontic notion and sketched a commitment based model for agent communication as commitment is usually seen as a directional act between two agents. This addressed the dynamic nature of obligation to model the fact that obligation can associate with to specific group or organization. An abstract form of commitment is described in the following:

$$\text{Commitment} =_{\text{Def}} C(x, y, [z], e)$$

e : action	x : committer
y : committee	z : associator, the role of witness

Fig. 2. Abstract of Commitment

Based on the definition, we define a number of speech acts which particularly associate the security policy objects of the communicating entities, such as promise, cancel, appoint and revoke. All the speech acts are relevant to security objects and agent communication. We will introduce the commitment based speech acts definition for each of them next. We follow Colombetti in using the term “committer” for the agent towards whom an obligation is assumed by a committee, although the action of committing is actually carried out by the committee and not by the committer.

Speech Acts are originally adopted as the basic units of communication among agents in Distributed Artificial Intelligence Communities. It also can be defined as the negotiation communications for policies specification in the large, distributed computing environments. In the case, we also can define a formalism based on the commitment formalization in the following:

$\text{SpeechAct} =_{\text{Def}} \text{SA}(x, y, z, e)$

[Searle 1975] classified speech acts into five categories: declarations, assertives, commissives, directives, and expressives. We will give the definition of some speech acts that affect the policy objects of the communicating entities in our policy language most, eg. Appointment, Delegation, Revocation and Promise.

3.1 Promise

Promise is a commissive act which is to commit oneself to execute of some actions for someone else under a limiting time constraint. The definition of this speech act, based on the abstract form of commitment is presented as following:

$\text{SA}(\text{Promise}) =_{\text{Def}} \text{Promise}(x, y, [z], \text{ActionObject}, \text{timeConstraints})$

- $\text{Promise}(x, y, [z], \text{ActionObject}) \vdash (\neg (x, y, [z], \text{ActObject}) \rightarrow V)$
- $\text{Promise}(x, y, [z], \text{ActionObject}, \text{timeConstraints}) \vdash ((x, y, [z], \text{ActObject}, \text{timeConstraints}) \wedge \text{timeConstraint} > \text{now}) \rightarrow V)$

The first promise rules represented that accomplish the promising iff x withdraw the promising, a violation is raised. The second rules shows the case that promising accomplishment is out of the time constraint.

3.2 Delegate

We refer the Delegation in our policy language here is the temporary appointment. It requests from an entity to execute some actions in a limiting time. When the commitment is declared, it will temporary persist until the time constraint reach the deadline.

$\text{SA}(\text{Delegate}) =_{\text{Def}} \text{Delegate}(x, y, \text{ActionObject}, \text{timeConstraints}, [\text{RevokeDel}])$

- $\text{Delegate}(x, y, \text{ActionObject}) \vdash \neg (x, \text{ActObject}) \rightarrow V$
- $\text{Delegate}(x, y, \text{ActionObject}, \text{timeConstraints}) \vdash ((x, y, \text{ActObject}, \text{timeConstraints}) \wedge \text{timeConstraint} > \text{now}) \rightarrow V$
- $\text{Delegate}(x, y, \text{ActionObject}, \text{timeConstraints}, \text{RevokeDel}) \vdash ((x, y, \text{ActObject}, \text{timeConstraints}) \wedge (\text{timeConstraint} < \text{now}) \wedge \text{RevokeDel}) \rightarrow V$

Delegation policy in our language is a temporary concepts. It requires that the delegator x must possess the right to be delegated. Otherwise it will raise a violation.

3.3 Appoint

The Appointment in our language is used for transferring the access rights and position in a certain domain. It is not necessary for an appointer X to possess the rights or in a position to be appointed.

$SA(Appoint) =_{\text{Def}} Appoint(x, y, ActionObject, [timeConstraints], [RevokeApp])$

- $Appoint(x, y, ActionObject, timeConstraints) \vdash ((x, y, ActObject, timeConstraints) \wedge timeConstraint > now) \rightarrow V$
- $Appoint(x, y, ActionObject, timeConstraints, RevokeApp) \vdash ((x, y, ActObject, timeConstraints) \wedge (timeConstraint < now) \wedge RevokeApp) \rightarrow V$

RevokeApp provides the cancellation of the correspondence appointment which a commitment for the reason of completed or updated. The consequences of a valid appointment, is created an obligation rule or a permission rule.

3.4 Revoke

Revoke is the action for the appointer to remove the rights or actions which is appointed to another subject.

$SA(RevokeDel) =_{\text{Def}} RevokeDel(x, y, ActionObject, [timeConstraints])$

- $Revoke(x, y, ActionObject, [timeConstraints]) \vdash \neg (x, ActObject) \rightarrow V$
- $Revoke(x, y, ActionObject, [timeConstraints]) \vdash (x, y, ActObject, timeConstraints) \wedge (timeConstraints > now) \rightarrow V$

$SA(RevokeApp) =_{\text{Def}} RevokeApp(x, y, ActionObject, [timeConstraints])$

- $RevokeApp(x, y, ActionObject) \vdash \neg (x, y, ActObject) \rightarrow V$
- $RevokeApp(x, y, ActionObject, timeConstraints) \vdash (x, y, ActObject) \wedge (timeConstraints > now) \rightarrow V$

The consequence of a revoke speech act is the appointee no longer possess the appointed rights or the delegated action is forbidden.

4 Conclusions

Here we gave a simple analysis of the relationship between commitment and agent speech acts. In the proposed formalism, an agent's commitment is considered as a special kind of action which can change its deontic preference and its obligation to doing some action is based on the preference and the effects of the action. We presented how obligations arise from commitments via updating of deontic preference. The integrated modeling approach shows it's possible to express and reasoning about the mutual relationship among these mental attitudes to express the security policy.

In this paper we primarily addressed a structure of deontic concept based policy modelling, and a commitment based formalism for speech acts of agent communication which we proposed to designed for authorization in mobile computing environment. But from the point of view of full mobile commitment management this just gives us a baseline, this work needs to be extended in the following directions:

- It is necessary to model the temporal aspects of commitments (in terms of elapsed time not just temporal ordering);
- and also needed to model the consequences of the violation atom being raised.

We will also conduct a comparison of our work with the approach of Abrahams and Bacon [2002], which resembles ours in recognizing that obligations are brought about by specific occurrences, but which differs from us in taking the position that “being obliged to do that which is forbidden is not a logical contradiction”. Abrahams and Bacon have also drawn attention to the importance of *Situation* in relation to specific obligations, and we shall investigate the extent to which the resources of *situation theory* [Devlin, 1991; Devlin & Rosenberg, 1996] can be brought to bear on these problems.

5 Practical Application

Our proposed practical application is in the distributed mobile computing environment. We aim to create a deontic concepts based policy structure language with which agents can interact, and methods that precisely and unambiguously provide meanings for agent communication. The framework is designed to be a general framework for future research of many varieties of agent based systems and supports a flexible security policy infrastructure for the dynamic, mobile computing environment.

The implementation will involve the use of mobile agents as autonomous intermediaries between nomadic users and mobile infrastructure services like laptop, mobile phone, PDA, et al. We use agents to manage some interactions for these distributed services. For instance, a user may require its agent to collect information about people present in a room and display this in a non-intrusive way. That collaborating software agents work autonomously to provide the ideal working environment while taking care to respect the cognitive load of the user who is handling many interactions with his colleagues.

The agents are designed to establish certain categories of speech acts via agent conversation. Our commitment based rules will catch these speech acts which coordinate the activities of multiple cooperative agents.

References

- Abrahams, A & J Bacon (2002) “The Life and Times of Identified, Situated and Conflicting Norms”. *Proc Sixth International Workshop on Deontic Logic in Computer Science (DEON’02)* London, May 2002, pp. 3–20.
- Ahn G, J & Sandhu, R (2000) “Role-based Authorization Constraints Specification,” *ACM Transactions on Information and Systems Security*, Volume 3, Number 4, November 2000.
- Bacon J, Lloyd M and Moody K (2001) “Translating Role-Based Access Control Policy within Context” *Policy 2001, Workshop on Policies for Distributed Systems and Networks*. Springer LNCS no 1995.

- Bacon, J, Moody, K & Yao, W (2002) "A model of Oasis role-based access control and its support for active security". *ACM Transactions on Information and System Security (TISSEC)* 5.4, pp. 492–540.
- Bell D. E. and LaPadula L. J. (1975) "Secure computer systems: Unified exposition and multics interpretation". *Technical Report ESC-TR-75-306*, The MITRE Corporation, Bedford, MA, USA, March 1975.
- Cholvy, Laurence and Frederic Cuppens, "Analyzing Consistency of Security Policies." In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE Press, 1997. p.103–112.
- Colombetti, M (2000) "A commitment-based approach to agent speech acts and conversations". *Proc. Workshop on Agent Languages and Communication Policies*, 4th International Conference on Autonomous Agents (Agents 2000), Barcelona (E), 21–29
- Devlin, K (1991) "Logic and Information". *Cambridge: Cambridge University Press*. 1991
- Devlin, K & Rosenberg, D (1996) "Language at Work: Analyzing Communication Breakdown in the Workplace to Inform Systems Design". *Stanford: CSLI*. 1996
- Haag, Z (2000) "Deontic Logic Based Process Modelling for Coordination Support in Virtual Software Corporations". *PhD Thesis*, Glasgow Caledonian University, June 2000.
- Haag, Z, Foley, R & Newman, J (1998) "A Deontic Formalism for Coordinating Software Development in Virtual Software Corporations". *IEEE 7th Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98)*, Stanford University, CA.. Los Alamitos, CA: IEEE Computer Society Press. pp. 15–20.
- Hayton, R, J Bacon, K Moody (1998) "OASIS: Access Control in an Open, Distributed Environment", *Proc IEEE Symposium on Security and Privacy*, Oakland CA, pp. 3–14, May 1998
- Meyer, J. -J.Ch. (1988) "A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic" in *Notre Dame Journal Of Formal Logic* 29(1), 1988 pp. 109–136
- Meyer, J. -J.Ch. & Wieringa, R.J. (ed) (1993) "Deontic Logic in Computer Science: Normative System Specification". *Chichester: John Wiley & Son*.
- NCSC (1987) "A Guide to understand Discretionary Access Control in Trusted Systems". *National Computer Security Center*, 1987.
- Ortalo, R. "A Flexible Method for Information System Security Policy Specification". In *Proceedings of 5th European Symposium on Research in Computer Security (ESORICS 98)*. 1998. Louvain-la-Neuve, Belgium, Springer-Verlag.
- Sandhu R S, Coyne E J, Feinstein H L & Youman C E (1996) "Role-Based Access Control Models" *Computer* Vol. 29 No 2 (Feb) pp. 38–47.
- Searle, J R. (1975) A Taxonomy of illocutionary acts. In K. Gunderson ed., *Language, mind, and knowledge* (Minnesota Studies in the Philosophy of Science VII), University of Minnesota Press, 344–369. Reprinted in J. R. Searle (1979), *Expression and Meaning*, Cambridge University Press, Cambridge, UK
- Sergot, M J (2001) "A computational theory of normative positions." *ACM Transactions on Computational Logic* 2(4): 581–622 (October 2001)
- Winograd, T & Flores F. (1987) "Understanding Computers and Cognition". *New York: Addison Wesley*.

Business to Consumer Markets on the Semantic Web

Robert Tolksdorf¹, Christian Bizer², Rainer Eckstein³, and Ralf Heese³

¹ Freie Universität Berlin, Institut für Informatik, AG Netzbasierte Informationssysteme,
Takustr. 9, D-14195 Berlin, Germany,
research@robert-tolksdorf.de
<http://www.robert-tolksdorf.de>

² Freie Universität Berlin, Freie Universität Berlin, Institut für Produktion,
Wirtschaftsinformatik und OR
Garystrasse 21, D-14195 Berlin, Germany
bizer@wiwiss.fu-berlin.de
<http://www.bizer.de>

³ Humboldt-Universität zu Berlin, Institut für Informatik, Datenbanken und
Informationssysteme, Unter den Linden 6, D-10099 Berlin, Germany,
{[Rainer.Eckstein](mailto:Rainer.Eckstein@dbis.informatik.hu-berlin.de),[rheese](mailto:rheese@dbis.informatik.hu-berlin.de)}@dbis.informatik.hu-berlin.de

Abstract. The emerging Semantic Web technologies have the potential to deeply influence the further development of the Internet Economy. In this paper we propose a scenario based research approach to predict the effects of these new technologies on electronic markets and business models of traders and intermediaries. We develop a concrete scenario for the domain of business to consumer electronic commerce and explain its technical realization. Furthermore, we analyse the potential impacts on market participants and highlight a range of technical and organizational obstacles which have to be solved before semantic technologies can be deployed in an e-business context.

1 Introduction

The past 10 years of Web evolution have established electronic markets and led to the rise and fall of the “new economy”. The next 10 years may be characterized by the transformation of the Web from a document publication medium intended for human consumption into a medium for intelligent knowledge exchange [14]. This development is led by the W3C Semantic Web initiative with a joined effort of scientific (MIT, Stanford, ILRT etc.) and business institutions (HP, IBM, Nokia etc.). The basic idea of the Semantic Web is to publish—in addition to classic HTML pages—data directly on the Web. The vision is to use the Web as a global distributed database, which could be queried like a local database today. The W3C Semantic Web architecture stack is defining the reference architecture in the ongoing standardization process. The standardization of the basic layers of the architecture is already at a very advanced state. The standardization of the higher layers (security, logic, proof, trust) is just starting. While there are still a lot of technological issues to be solved, the Semantic Web is in a phase in which consensus about its potentials exists.

However, it is unclear what economic effects these new technologies and their applications will have on markets and enterprises:

- How do semantic technologies affect market-transparency in electronic markets?
- How does this affect the business models of market participants and intermediaries?
- How does an increased efficiency of information exchange affect industry specific value chains?

Even if the vision of a global distributed database maintained on a peer-to-peer basis does not become reality in the midterm, Semantic Web technologies provide a huge longterm potential in several application domains [8]:

- *Enterprise Information Discovery and Integration.* Ontology based data integration frameworks will significantly reduce integration costs. Seen from a system architecture perspective, a stronger decoupling of data and applications will become possible. Data could become an independent resource, used by several applications. Semantic Web technologies could also play a major role in the Web Service description, discovery, composition context. Ontology based service description frameworks could push the possibilities of automatically combining services offered by different service providers another step forward [2].
- *E-Commerce.* The development of XML-based e-commerce standards has led to the same problems EDI initiatives ran into in the past: A wide range of competing standards has been developed and is used by different trading communities. Today suppliers often have to maintain their product catalogs in several XML formats for different market places and customers. Semantic technologies could help to solve these problems by offering a framework for standard mapping and to identify entities like products and traders across different standards. Using RDF's URI identification mechanism, the integration of different product descriptions, ratings, and offers from multiple independent sources by a shopping portal or single user agent might become possible. This would enlarge the data basis of shopping and comparison agents and enhance their usefulness. Seen from the market perspective, this could lead to an increase in market transparency and make the business models of a range of trading intermediaries and market places obsolete.
- *Knowledge Management.* Ontology based search engines will offer a direct access to information sources in corporate intranets and on the global Web, which will reduce search costs. In addition, adaptive Web sites could enable a dynamic reconfiguration according to user profiles, and make precisely the information needed available in the personally desired format and on the preferred device. Because semantic knowledge networks are based on language independent ontological concepts, it could be even possible to render a large amount of Web content in the user's preferred language.

Before these potentials can be realized, there is still a range of technical and organizational issues to be solved. There has been an agreement on the basic system architecture, data formats and protocols. In order to integrate data from different sources, there has to be consensus about a set of domain ontologies and mappings between them. Mechanisms to decide which data is trustworthy have to be developed and a relevant amount of data has to be published according to the RDF data model and the ontological vocabularies.

The access to distributed, machine readable and semantically annotated information will widely influence the further development of the Internet economy, if these problems can be solved.

In the following section we propose a scenario based research approach to predict the effects of Semantic Web technologies on electronic markets. In section 3 and 4 we develop an e-commerce scenario based on Semantic Web technologies for the example domain of car accessories. First we explain the current state of technological development in that domain and highlight the major shortcomings. Afterwards we describe the potentials of Semantic Web technologies to solve these problems. In section 5 we analyse the potential impacts on market transparency and the business models of the market participants.

2 Scenarios for the Future Semantic Web

Taking up ideas from [16], we use scenarios as a basis to explore the effects of the deployment of Semantic Web technologies. A scenario is defined in the cited work as “a possible set of events that might reasonably take place” in the future. Scenarios should stimulate thinking in order to enable a management of change. The change which we foresee is a shift from the representation of documents on the Web to data and information in the Semantic Web. The interesting questions from a business perspective are how this change affects current market situations and business models.

Jarke in [16] expands the above definition further into “A scenario is a description of the world, in a context and for a purpose, focusing on task interaction. It is intended as a means of communication among stakeholders, and to constrain requirements engineering from one or more viewpoints (usually not complete, not consistent, and not formal)”. Our work will use scenarios to assess the economic potential and effects of the Semantic Web and knowledge nets in value chains.

With our work, we approach the impact of semantic technologies from the business and the technical viewpoint, in order to make predictions about the influence of the new technologies on markets, enterprises and individuals. The results will be the mentioned communication mean amongst stakeholders, but also requirements that might be used in the development of further Semantic Web standards.

We derive our scenarios from two subscenarios. One is a technological scenario that assumes the availability of semantic technologies in a perspective of several years. The second is a deployment scenario which assumes that semantic application specific knowledge is available in machine readable form usable for applications. From the combination of these two projections we derive e-business scenarios, for analysis and experimentation as shown in figure 1. Each such scenario is a projection of today's developments or a forecast of the state of the Semantic Web in five years.

In the current early stage of development this approach allows us to make statements about the implications of the new technology on the participants of the scenario. Who will benefit? Who will lose market positions or will have to change his business model? This analysis of the participant's roles combined with the analysis of technical restrictions allows us to project if a scenario has chances to be realized.

Every scenario will be relative to an application domain. In the domain of electronic commerce, we think that Semantic Web technologies will have different impacts depending on the characteristics of the specific markets. Semantic Web technologies are likely to have the greatest impact if a market fulfills the following set of criteria:

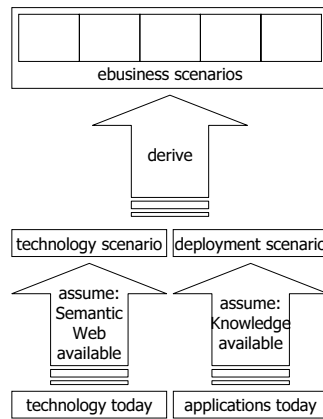


Fig. 1. The scenario based approach

- The application domain must possess a market with multiple suppliers. Integrating data from multiple sources and comparing products and services on a semantic basis are the key capabilities that we expect from the usage of Semantic Web technologies. To reason about its influence, there must be enough sources of products and services that can be compared.
- The suppliers must have similar influence on the market and equal access to it, since in a market without sufficient competition no evolutionary changes can be expected.
- The products and services traded must be uniquely and precisely identifiable, because the Semantic Web relies on the unique identification of things about which statements are made.
- The products and services should already be traded via the Internet, so that customers are already used to this distribution channel.
- To keep things simple, there should be no negotiations involved in a usual business process and the products should be traded at fixed prices.

Following these criteria, consumer markets where medium complex, high involvement products, like video cameras, HiFi, computer or other electronic equipment or car accessories are traded, could benefit most from the usage of Semantic Web technologies. Examples of service markets which could benefit are the travel and the job market. As an example for a first scenario following our approach, we select the car accessories market. It is described in the next section.

3 Current State of the Domain Car Accessories

In order to identify potentials for improving electronic commerce by the use of Semantic Web technologies, this section describes the current state of development in an example domain and highlights weaknesses of the technologies currently in use.

We have examined several online stores which offer car accessories and paid special attention to the way products were presented, classified and identified. As a second focus,

we evaluated which sites offer further information relevant to a purchasing decision, how difficult it is to find these information sources and to compare the information about a specific product. We use the example of buying tires to illustrate the different aspects.

The Web sites relevant for our study can be classified into the following groups based on assortment strategy, business models and kinds of information provided (sample web sites in parentheses):

- *Online stores* offering a wide range of products, including a shallow assortment of car accessories, like child safety seats, cleaning supplies or roof racks. (*www.quelle.de*, *www.neckermann.de*)
- *Online stores* with specialized, deep assortments, offering a wide range of tires or just high performance tires as part of a sports car assortment. (*www.tirerack.com*, *www.discounttire.com*)
- *Electronic marketplaces and online auctions*, where individuals and companies offer new and used tires. (*www.ebay.com*, *auctions.yahoo.com*)
- *Specialized search services* for car accessories which lead to both online and offline stores.
- Web sites of *manufacturers* describing their products. (*www.goodyear.com*, *www.dunloptire.com*)
- Web sites of *car magazines* offering product test and comparisons of different tire types. (*www.caranddriver.com*, *www.europeancarweb.com*)
- *Rating sites* like *epinions.com*, where consumers report experiences with a product.

These sources together offer a huge amount of information about a specific tire and a wide range of possibilities to purchase it. The problem for the customer is to find all relevant information sources and to compare the information fragments offered. This problem is aggravated by the fact, that most content of the Web sites is generated dynamically from databases and therefore not indexed by search engines like Google.

Examining the product descriptions on the various stores we noticed that the structure does not diverge much. The main description elements are item short text, detailed item description and price of the item. In the case of tires the description contains at least the brand, the name and the size of the tire. The notation of the size is standardized. For free text item descriptions some sites use a kind of controlled vocabulary, e.g. enhanced wet, dry and snow traction, and differ often in small details. For instance some provide additional information like illustrations or handling characteristics of tires.

The stores usually use their own methods for enumerating products. Therefore different online stores assigned different item numbers to the same product. Many shops do not publish their internal item numbers and use the item short text as unique identifier on their pages instead. The stores examined mostly use a combination of the brand (Goodyear), name (Eagle F1 GS EMT) and size (P245/45ZR-17) for identification purposes.

All shops categorize their items in generalization-specialization hierarchy which is used for navigating through the offers. The average categorization depth is three levels. Different stores use different hierarchies. In our tire example these hierarchies are based on criteria like brand, size or style.

In conclusion, someone searching for car accessories is confronted with a wide range of information sources offering separate pieces of information about an item and a variety

of shops, marketplaces and actions offering the item itself. The main problems using current Web technologies are:

- *Finding* all relevant information sources and online stores for a specific product.
- *Integrating* all information available on the Web, for comparing products and vendors.

Thus buying decisions today are based only on parts of the relevant information available on the Web.

4 Deploying the Semantic Web in the Car Accessories Domain

Semantic Web technologies address the above two issues directly and improve market transparency. The following sections describe how the technologies could be utilized. After an architectural overview we present two typical use cases which show the benefits of the architecture. Next we describe the ontology development, the information provision and the information usage in more detail.

4.1 Architectural Overview

The Semantic Web technologies offer three important building blocks for our e-commerce scenario [18]:

- The use of *URIs as a global identification mechanism* for products and traders.
- The *RDF data model* together with its XML-based serialization syntax for the direct publication of data on the Web.
- The *Web Ontology Language (OWL)* for the definition of common terms and concepts needed to understand RDF data.

In an e-commerce scenario based on Semantic Web technologies, the market participants will have the following roles.

A manufacturer of a product will define a URI for his product which identifies it globally, e.g. http://goodyear.com/eagle_f1/gs.emt/p245/45Zr-17. Based on this URI he will publish product descriptions and additional technical information about the product using the RDF data format. A merchant offering the product then only has to publish his price for the product together with shipment and payment details, referring to the product with the URI. The use of a common identification schema and a common data model will allow shopping portals to integrate all information available on the Web about the item.

The portal will not collect links to Web pages or online shops, it will directly collect the data from the sites. This allows the integration and direct comparison of the content of different sites. A car accessories portal would collect all available information about an item from manufacturers, merchants, test and rating services and integrate them into a personalized offer for customers. Buyers then can use one central portal instead of collecting information fragments from different sites. Under the assumptions that all relevant information providers participate and that mappings between different description schemata are possible, large parts of the information in the market would be available in an open and machine processable manner. Figure 2 illustrates the role model for the scenario and gives an overview about the technical architecture.

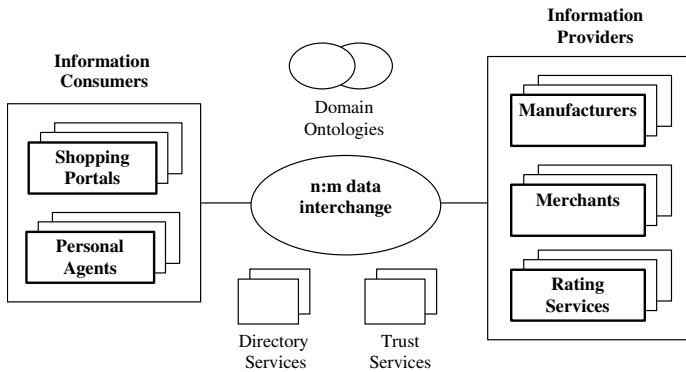


Fig. 2. Architecture Overview

4.2 Use Cases

The following two use cases can illustrate the benefits a shopping portal based on Semantic Web technologies provides for the user. In the first a customer wants to order a particular set of tires knowing the brand (Goodyear), the name (Eagle F1 GS EMT) and the size (P245/45ZR-17). A typical question of interest is: Which shop sells the particular tires for the “best” price and with the “best” conditions? What “best” means depends on the requirements and preferences of the customer.

The customer could also want to know whether there are similar tires in the market which are better or cheaper than a given one. By that, the customer wants to search per article and not per shop. Without Semantic Web technologies the customer has to visit a few stores known to him manually to gather the needed information. Not only is this procedure very time consuming, but it is also unlikely that he eventually finds the best offer available. This is due to different navigation paths, different search mechanisms and different product descriptions in the stores.

Standard search engines are no good either help, since they are unable to search on the *concepts* that describe articles, but only on the words used for naming them. A portal based on Semantic Web technology would have a near global view at the market. It would know nearly all shops and their offers, the manufacturers and their products, and value-added services like rating services. The portal can take the semantic descriptions of the articles and match them with the requirements of the customer.

In the second use case a customer only has a vague picture of the product he wants to buy, e.g. winter tires. The customer knows his car model and finds the allowed tire sizes in his vehicle documents. But there are additional variable properties, for which the customer often does not know the precise technical terms. Here, the customer wants information search per product group instead of search per shop.

For example, the customer needs tires that are safe both at high speed driving and heavy rain. The properties “high speed” and “very good rain behavior” have to be named correctly to formulate a specific query with a standard search engine. It is, however, hardly possible to name every potential member of a product group in such a query.

Using a portal based on Semantic Web technologies, the knowledge included in the domain ontologies could be used to assist the customer to narrow the conceptualization of his vague requirements into the technical terms or specific concepts. Again semantic descriptions and suitable search algorithms would lead the customer to the best matching products.

4.3 Ontology Development

The communication between the market participants is based on a set of ontologies which provide shared terms to describe products, traders, shipment and payment options. Using the tire example again, the market ontology will include the concept of a tire and define several properties like size or rain behavior together with the range of possible property values. The market ontology is the result from the merging of existing ontologies by defining mappings between them.

To create the ontologies needed for the market, a lot of standardization efforts from the EDI community can be reused. Standards like EDIFACT or ebXML define already many of the necessary business concepts and can be converted into ontologies. Examples of such reuses are the RDF versions of the eClass [6] and UNSPSC [17] product classification schema. An overview of the relevant standards can be found in [7].

The existing standards provide the general framework, but have to be complemented with more fine grained domain concepts. Because the manual creation of fine grained ontologies is very costly, different semi-automatic approaches for ontology creation using text mining and language processing tools are being researched. In our example domain, many concepts and relations between them could be extracted from the existing Web sites and product descriptions. An overview of the different approaches and ontology creation tools can be found in [11].

The experiences from the EDI community show that it is impossible to reach agreement on a single standard for a domain. The Semantic Web approaches this problem by allowing the co-existence and co-usage of multiple ontologies for the same domain. To integrate them, mappings amongst similar or equal concepts in separate ontologies can be defined. In perspective, these mappings lead to ontology convergence. For example the technical term “wet traction” is related to the colloquial superconcept “rain behavior”. There are different manual and semi-automatic approaches being researched [1,19] to generate mappings between different ontologies.

4.4 Information Provision and Usage

To participate in an electronic market based on Semantic Web technologies, the information providers (producers, merchants and rating services) will map their local data models and identification schemata to an ontology used in the market. Collecting these mappings a network of ontologies is established.

Most product and pricing data is stored today in relational databases and can be easily reused on the Semantic Web. There are different approaches to map relational data into RDF [15,4]. After the mapping, the RDF data is published using standard Web servers like Apache or specialized query interfaces as in [12,22].

All the published information will be accessed by potential buyers using either personal agents who collect information for them or using semantic shopping portals which provide the access to the information through a standard HTML interface. A shopping portal which is presenting the published data to the customers will:

- Use a directory service to locate information providers for the car accessories domain.
- Use a robot to collect the data from the different providers.
- Decide which data is trustworthy and shall be used in the portal.
- Use a mapping engine to integrate data published using different ontologies.
- Render the data according to the user's preferences (level of detail, device, language, ...).
- Provide semantic search services for the customers based on the knowledge included in the ontologies and the available market data [13].

To decide about the trustworthiness of data found on the Semantic Web the portal would use trader independent trust services. These trust services could use trust mechanisms based on information source, context or the authors reputation [21]. There are different technical approaches for a Semantic Web trust architecture being discussed in [10] and [5] using digital signatures together with Web of Trust mechanisms.

Following the above architecture, a portal would have to store huge amounts of collected RDF data and provide easy access to this data for his customers. An overview of specialized databases and query languages to accomplish this task can be found in [3] and [20].

To allow searches on a semantic level [13], the shopping portal will use a reasoning engine which combines the knowledge included in the ontologies with the instance data collected from the Web. Thus vague, similar, or synonym concepts can be matched against the existing information. For example the vague concept "fast" could be matched to a special tire property.

5 Scenario Analysis

Our scenario is based on the new technological foundation defined by Semantic Web standards and raises a couple of interesting economic questions. This section will discuss the possible implications on the business models of the participants and point to open issues, which have to be solved before the technologies can be deployed with success in real markets.

The communication processes between manufacturer, merchant and shopping portal are changed in the Semantic Web scenario. Formerly there were controlled 1:n communication links between them. The manufacturers supplied a known set of merchants with product information. The merchants published this information on their Web sites and forwarded it to the set of marketplaces and auctions in which they participate. Semantic Web technology changes this communication structure. An offer published by a merchant can be used by an unknown number of market places. Formerly controlled 1:n communication links change to an open n:m communication situation.

There are also changes in the kind of information the different parties have to provide. Classically, product description were provided and maintained redundant by every single shop or market place. In the Semantic Web Scenario a merchant would not have to replicate product descriptions to his site, because they are already available from the manufacturer and can be integrated with his offer by the shopping portal.

The application architecture and the possibility of higher market transparency arise new strategic questions for the market participants which will ultimately determine their willingness to participate: Is the architecture in the economic interest of all potential participants? Is it possible to construct an economic win-win situation which would motivate all parties to participate?

For the *customers*, these questions can easily be answered positively. Customers would benefit from the higher market transparency and could make their buying decisions based on a solid, computable information basis. For the *manufacturers* the answers are also positively. They are interested in informing as many potential buyers as possible about the existence of their products. A second advantage for the manufacturers is that they gain more control on the information presented to the customers about their products.

For merchants and shopping portal operators, it is unclear whether the questions can be answered positively. On the one hand it would be much easier to find *merchants* for a specific product if the product is globally identified by an URI. Merchants would also save costs, because publishing data on the Semantic Web is cheaper than maintaining high quality human-readable online stores. But what is likely to weigh more is the fact that merchants are not interested in high market transparency, because in a perfect market their profit margins fall close to zero. Their information advantage would decrease, allowing them to differentiate themselves from other merchants only by additional services.

Shopping portal operators face similar problems. They would all operate on the same information basis. This would allow them to differentiate only in the way they present the information to potential buyers and by additional services like insurances and bonus programs.

Another question is the business model of the *rating services* in the scenario. Today rating services in consumer markets like epinions.com make their profits and are able to pay consumers for their ratings by the revenues they are marking from advertisements on their Web sites and by provisions they are getting from the merchants when a buyer is directed to a shop by the rating service. If the rating data would be published on the Semantic Web and presented to the buyers together with the product information by the shopping portals, the business model of the rating services would have to change to a pure pay-per-view model.

Before Semantic Web technologies can be successfully used in an e-business context and the scenario described above could be realized, numerous technical and organizational obstacles have to be overcome.

- *Missing ontologies.* Most of the RDF ontologies available for the domain of e-commerce are just research prototypes and hardly fulfill the requirements of real electronic markets. There are a lot of standards, like ebXML or EDIFACT, which could be used on semantic networks. But there is no awareness for potentials of Semantic Web technologies in the communities developing these standards.

- *Missing Identification Schemes.* There are also no commonly accepted identification schema for products, companies, places and people within the Semantic Web community. For integrating information from different sources commonly accepted URI schemata or at least URI mapping mechanisms would be necessary. The identification problem has also been addressed in the EDI community for a long time. So existing approaches like EAN numbers for products, Dun and Bradstreet numbers for companies or UNCODOCODES for places could be reused. What would have to be achieved is a consensus about a set of identification schema and ways to publicize product and organizational URIs. If the the Semantic Web grows in the future, it could be common practice to have URIs on business paper and product labels, like it is common practice today with URLs.
- *Security and Trust.* In a distributed environment it is important to know who made which assertion in which context. In our scenario a central enabling factor is the question, whether it is possible to build trust mechanisms which allow shopping portals and customers to decide which data published on the Semantic Web is trustworthy and should be used in buying decisions. When the Semantic Web starts to have an economic impact it is likely that a lot of intended misinformation, for example about competitors, is published and that there are many trust problems arising from the “everybody can say everything about everybody” paradigm. One example of these problems is the question what should happen with true information a merchant publicizes about his more expensive competitor which is not publishing the information himself. It would be useful for a shopping portal to use this information, but hard to discriminate it from false information published elsewhere. Trust and security mechanisms are part of the W3C Semantic Web layer cake, but have to be matured by more research before they can be used in an open e-commerce context.
- *Privacy.* The new possibilities to integrate data from different sources promised by semantic technologies also entail new threats to privacy. The RDF data published on the Web will form an El Dorado for market researchers. Many of the new technical possibilities seem to be illegal with respect to national privacy legislation like the German Datenschutzgesetz.
- *Diffusion strategies.* In order to show the potentials of the Semantic Web and to involve larger communities, seeding application and business cases with clear economic benefits are needed. The Semantic Web is facing a chicken/egg problem today. Because there is hardly any real world RDF data online [9] it is difficult to demonstrate the benefits of the technology. Because the benefits are unclear, major potential users remain in a waiting position and do not publish RDF data online.

6 Conclusion

In this paper we developed an e-commerce scenario, showing how semantic technologies could influence electronic markets. We described the potentials of the new technologies and the open issues which have to be solved before the scenario can be realized.

Our scenario-based approach is a realistic tool to project today’s technology and the deployment of the Semantic Web into a future of about five years. The work presented here is a first step in a series of deeper analysis and experimentations.

The analysis will study the economic consequences of our scenarios on the participants. Experiments will assess, how realistic the technological assumptions are, based on current technologies, and how useful the projected results will be for the deployment context.

The analysis will lead to statements about changed characteristics of markets and value chains. The experiments will showcase future technologies and derive requirements on the future technological development, which then could influence the current standardization processes.

To realize the experiments, we will combine existing software components with our own developments. In the course of our future work this will lead to an evaluation kit for the simulation of our scenarios. The work will be performed in the project *Wissensnetze* funded by the German Ministry of Research BMBF as part of the Berlin Research Centre for the Internet Economy InterVal.

References

1. K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. The Chatty Web: Emergent Semantics Through Gossiping. In *Twelfth International World Wide Web Conference, WWW2003*, 2003.
2. A. Ankolekar et al. DAML-S: Web Service Description for the Semantic Web. In *The First International Semantic Web Conference (ISWC)*, 2002.
<http://www.daml.org/services/ISWC2002-DAMLS.pdf>.
3. D. Beckett. Scalability and Storage: Survey of Free Software / Open Source RDF storage systems, 2003.
http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report/.
4. C. Bizer. D2R MAP - A Database to RDF Mapping Language, 2003.
<http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rmap/www2003-D2R-Map.pdf>.
5. C. Bizer. Semantic Web Trust and Security Resource Guide, 2003.
<http://www.wiwiss.fu-berlin.de/suhl/bizer/SWTSGuide>.
6. C. Bizer and J. Wolk. RDF Version of the eClass 4.1 Product Classification Schema, 2003.
<http://www.wiwiss.fu-berlin.de/suhl/bizer/ecommerce/eClass-4.1.rdf>.
7. Diffuse IST Project. Standards and Specifications List, 2003.
<http://www.diffuse.org/standards.html>.
8. Y. Ding et al. The Semantic Web: from Concept to Percept, 2003. To appear in OGAI.
9. A. Eberhart. Survey of RDF data on the Web, 2002.
<http://www.i-u.de/schools/eberhart/rdf/rdf-survey.pdf>.
10. J. Golbeck, B. Parsia, and J. Hendler. Trust Networks on the Semantic Web. In *Twelfth International World Wide Web Conference, WWW2003*, 2003.
11. A. Gómez-Pérez et al. A survey on ontology tools, 2002.
www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf.
12. R. Guha and R. McCool. TAP: A Semantic Web Platform, 2003.
<http://tap.stanford.edu/tap.pdf>.
13. R. Guha, R. McCool, and E. Miller. Semantic Search. In *Twelfth International World Wide Web Conference, WWW2003*, 2003.
14. J. Handler, T. Berners-Lee, and E. Miller. Integrating Applications on the Semantic Web. In *Journal of the Institute of Electronic Engineers of Japan*, 2002.
15. S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *Twelfth International World Wide Web Conference, WWW2003*, 2003.

16. M. Jarke, X. T. Bui, and J. M. Carroll. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal*, 3(3–4): 155–173, 1998.
17. M. Klein. DAML+OIL and RDF Schema representation of UNSPSC, 2002. <http://www.cs.vu.nl/~mcaklein/unspsc/>.
18. M.-R. Koivunen and E. Miller. W3C Semantic Web Activity, 2001. <http://www.w3.org/2001/12/semweb-fin/w3csw>.
19. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies. In *Twelfth International World Wide Web Conference, WWW2003*, 2003.
20. L. Miller. Databases, Query, API, Interfaces report on Query languages, 2003. http://www.w3.org/2001/sw/Europe/reports/rdf_ql_comparison_report/.
21. P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation Systems. *Communications of the ACM*, 43(12): 45–48, dec 2000.
22. A. Seaborne. Joseki - Project Homepage, 2003. <http://www.joseki.org>.

Modeling Business Collaborations in Context

Birgit Hofreiter and Christian Huemer

Department of Computer Science and Business Informatics
University of Vienna, Liebiggasse 4, 1010 Vienna, Austria
{birgit.hofreiter,christian.huemer}@univie.ac.at

Abstract. Standard e-business document types are usually too ambiguous due to an overwhelming choice of data elements. Business partners must agree on a shared subset and adapt their application interfaces accordingly. Small and medium enterprises (SMEs) cannot perform this task. UN/CEFACT's Modeling Methodology (UMM) provides a methodology to define unambiguous business collaborations allowing software vendors to integrate corresponding APIs into their business software. Business collaborations depend on their business context, i.e. parameters describing the business environment. Instead of developing different models for each specific business environment, we need a single model for a specific business goal clearly specifying the context variations. In this paper we extend UMM to show how a generic business collaboration is adapted to different business contexts. This is demonstrated by a case study on ordering/selling books as well as tourism products.

1 Introduction

Business-to-business e-Commerce (B2B) is not particularly new. For a long time the field was dominated by electronic data interchange (EDI) standards [6]. However, small and medium enterprises (SMEs) failed to implement EDI due to the complexity of EDI standards, as UN/EDIFACT or X12. XML seemed to be the solution to traditional EDI problems. However, the most important limitation still exists [9]: Standard document types are overloaded. Usually, a specific partnership requires about 3% of the data elements of a document type. Thus, business partners have to agree on a corresponding subset and additional rules. Each partner must implement a mapping between the in-house system and the exchange format that follows the agreement.

This paper is based on the following premises: SMEs will never be able to implement this type of mapping. Consequently, SMEs need software packages that provide both the business functions needed (e.g. purchase order handling) and the B2B functionality (e.g. ordering goods from a business partner). Only if these functions go hand in hand together in low cost commercial off-the-shelf software (COTS), we will see B2B e-commerce to take off. This requires a paradigm shift for producing B2B standard specifications. These specifications must unambiguously describe business processes among two or more partners, which we call business collaborations. Only unambiguous definitions allow software vendors to integrate these collaborations into their software products. Integrating a set of APIs requires both unambiguous data structures and an unambiguous choreography. The choreography defines the dynamic

aspects of the collaboration in terms of agreed information flows, whereby each exchange leads to an agreed business state change.

A methodology similar to a software process is needed to develop unambiguous business collaborations. For this purpose, the United Nation's Centre for Trade Facilitation and e-Business (UN/CEFACT) has developed UN/CEFACT's Modeling Methodology (UMM) [15,8], which is based on the Unified Modeling Language (UML) [1]. It describes a method and supporting components to capture business process knowledge, independent of the underlying implemented technology so that the business acumen is retained and usable over generations of implemented technology.

Having been involved in the development of UMM since day one, we present its basic concepts and strengths. However, UMM is an ongoing effort that still needs improvements. Its current version 12 perfectly guides through the process in case of a very specific business goal in a detailed business environment. This would lead to a multitude of models. Instead of developing different models for each specific business environment, we need a single model for a specific business goal clearly specifying the context variations of the business environments. Version 12 does not specify a methodology for reusing business objects to assemble the information exchanged. Furthermore, we are still missing an example which spans all over the UMM process steps. Hence, this paper adds the following aspects to the current version of UMM:

- Binding context drivers to each model. Context drivers represent a set of parameters to describe the business environment. This concept ensures the specification of generic business models with rules to adapt to specific business environments.
- Assembly of information exchanged based on ebXML core components.
- A case study for a simple punch-out catalog that spans over all UMM steps and adapts to two different business environments.

The remainder of this paper is structured as follows: Section 2 presents an overview of other approaches considering business processes in a B2B environment. Furthermore, it motivates the use of an object-oriented approach and UMM in special. An introduction into the process steps of UMM is presented in Section 3. The case study of a punch-out catalog for book and tourism products in Section 4 follows these steps. A summary in Section 5 concludes the paper.

2 Related Work

The idea of standard business scenarios and the necessary services to support them was first created by the Open-edi reference model that became an ISO standard in 1997 [10]. Thereby Open-edi separates the "what" in the Business Operational View (BOV) from the "how" in the Functional Service View (FSV). The BOV covers the business aspects such as business information, business conventions, agreements and rules among organizations. The FSV deals with information technology aspects supporting the execution of business transactions.

Since Open-edi is not an implementation specification, projects started to deliver "Open-edi"-compliant implementations. The InterProcs system [11] is a prototyping environment to support the design and execution of Open-edi scenarios. The language used to represent these trade scenarios is based on petri nets. Petri nets provide a well known formalism to model and execute workflows. Since B2B always involves a type

of workflow between organizations, petri nets are appropriate to model these inter-organizational workflows. This concept is also used by other authors (cf. [18,19,13,12]). Although petri nets are well suited for modeling an unambiguous choreography for business collaborations, they do not model the structure of the information exchanged.

The object-oriented approach cares about both the statics and dynamics of a system. UN/CEFACT's Techniques and Methodologies Group (TMG) is developing an object-oriented methodology to define the BOV layer of Open-edi. The work started off in 1998, but the UMM spec is considered as a living document with multiple revisions. UN/CEFACT's, which is known for their UN/EDIFACT standards, builds their next generation of EDI on top of UMM. When UN/CEFACT and OASIS started the ebXML initiative, it was UN/CEFACT's vision UMM is used to create BOV standards and XML is used as key concept in the FSV layer. Accordingly, UMM is ebXML's modeling methodology, but it is not a mandatory part of ebXML (cf. [4]). Especially, during the 18-month initiative of ebXML, UMM made a lot of progress. TMG and ebXML members, such as SWIFT, TM Forum, EAN*UCC, and RosettaNet participated in the development. In 2000 the copyrights of the Business Collaboraton Framework (BCF), used by RosettaNet were transferred to UN/CEFACT and the BCF was merged into UMM's version 10. The ebXML core components specification includes the concept of customizing generic components to the special needs of a business context. This concept does not exist in UMM version 12. In the following sections we describe how to merge this concept into UMM.

Since UMM stops at the BOV layer, a transformation to an IT solution on the FSV layer is required (c.f. [2]). Today the IT solution of choice is definitely web services. The web service community with their work on web service choreography represents another group focusing on business processes in B2B. However, their conceptual work is strongly intermingled with the XML representation languages (e.g. BPEL, WSFL, WSCL, WSCI) that are of their primary interest [20]. Currently, there only exists a project proposal for a conceptual model for developing and describing web services and their composition [5]. Since UMM is independent of the underlying technology, it is as candidate to fill this gap. For this purpose mappings between UMM and the various web service languages must be developed, which is out of scope of this paper.

3 The UMM Process

UMM is a UML-based methodology for describing Open-edi scenarios. It concentrates on the business semantics of the BOV of Open-edi. The steps in UMM are similar to a software development process. Since it does not (yet) focus on the FSV of Open-edi, steps from the implementation onwards are not considered. Hence, UMM provides the business logic to develop middleware for B2B as well as software components to interface the B2B processes. In the following subsections we describe the steps of UMM. Each step is based on a meta model, i.e. a UML profile for UMM defining a set of stereotypes with tagged values and constraints [16]. We recommend to read this paper together with the UMM profile mentioned above.

3.1 Business Domain View

The first workflow of UMM is used to gather existing knowledge. It identifies the business processes in the domain of the business problems that are important to stakeholders. It is important at this stage that business processes are not constructed, but discovered. Stakeholders might describe intra-organizational as well as inter-organizational business processes. Both are valid and recorded. However, the description concentrates on so-called business interface tasks, where a business communicates with its partners. All the discovered business processes are classified according to a pre-defined classification schema. The final result of the business domain view allows a business process analyst to find opportunities for business collaborations that are constructed in the following workflows.

3.2 Business Requirements View

The goal of the business requirements view is to identify possible business collaborations in the considered domain and to detail the requirements of these collaborations. Business collaborations span over multiple business processes discovered in the previous workflow. Thus, a use case for a business collaboration must consider the views of different stakeholders. The description of the use case must present an harmonized view on the business collaboration being developed.

The business goals as described in the business collaboration use case description are realized by a business collaboration. In the current version of the UMM meta model there exists a 1:1 relationship between a business collaboration use case and its realization in the business collaboration. We suggest a 1:n relationship: A business collaboration use case will describe a generic business goal independent of the business environment. The realizations in the business collaborations are dependent of the business context of its environment. It seems to be straight forward that a generic business collaboration is realized in different business contexts (see Fig. 2 of the example). The business environment for each business collaboration must be documented. It is described by a set of tagged values representing context drivers. Candidates for these context drivers were identified during the work on ebXML core components [17]: business collaboration, business transaction, product classification, industry classification, geopolitical context, official constraint, business process role, supporting role and system capabilities.

3.3 Business Transaction View

The Business Transaction View represents the view of the business process analyst. In the first step the business collaboration is modeled according to the corresponding use case description. For each identified business collaboration a corresponding choreography, the so-called business collaboration protocol is modeled by an activity diagram (see Fig. 3). A business collaboration protocol designs one or more business transaction activities. A business collaboration protocol is not a transaction and is used in cases where a transaction rollback is inappropriate. For each business transaction the maximum performance time is documented. If this time is exceeded, the initiating partner has to send a failure notice. Furthermore, for each business transaction activity

it is defined whether or not more than one business transaction can be open at one time.

Each business collaboration protocol defines a choreography in a given business context. Thus, we recommend to add context drivers to the business collaboration protocol as tagged values. A business collaboration protocol might be generic, i.e. valid in multiple business contexts. However, some of its business transaction activities and transitions might be valid in a limited context. In this case context drivers are assigned to the corresponding business transaction activities or transitions (see Fig. 5 of the example).

The next step in the Business Transaction View is to detail each business transaction activity by a separate activity graph called a business transaction (see Fig. 6 of the example). Again the business environment is added to the business transaction as tagged values. The specification of business transactions builds up on the experience gained from the development of RosettaNet Partner Interface Processes (PIPs). A business transaction is made up of a requesting business activity performed by the initiating partner and a responding business activity performed by the responding business partner. The requesting business activity outputs a business document (represented by a object flow state) that is input to the responding business activity. A business document created by the responding business activity and returned to the initiating business activity is optional. A business transaction follows one out of six different types of patterns.

By analyzing the business transaction it is important to define the following values for both requesting and responding activities: Time to Acknowledge Receipt, Time to Acknowledge Acceptance, Time to Perform, Authorization Required, and Non-Repudiation of Origin and Content. The values for Non-Repudiation of Receipt and for Recurrence are defined only for the requesting business activity. Note, acknowledge of receipt is sent after grammar validation, sequence validation, and schema validation. Acknowledge of acceptance is sent after an additional content validation. Recurrence is the number of retries in case of control failures. The different types of business transaction patterns also differ in the default values for the parameters mentioned above.

Each business document exchanged in the business transactions is modeled in a class diagram. The following flags are set for each class and/or attribute: is confidential, is tamper proof and is authenticated. In order to guarantee reusability, the business documents must be built by common business objects. Unfortunately, the current version of UMM does not reflect this requirement. The meta model only defines that the business information exchanged is built by recursively structured information entities. We recommend that these business objects are built on the basis of ebXML core components [17]. In this specification a core component is defined as *“a building block that contains pieces of business information that belong to a single concept. Core components are characterized by the fact that they appear in many different circumstances of business information and in many different areas of business.”* Aggregate core components become business objects and basic core components their attributes. The type of an attribute corresponds to the type of the core component, which covers a content component and complementary components. When using a core component in an interchange, one must set it into the right business context - it becomes a so-called business information entity. In other words the core component is customized according to the business environment. Similarly, instead of simply reusing ambiguous business objects, an unambiguous customization is defined by setting

the context drivers. Thus, software providers will not face an overwhelming choice of components, but must support the customizations for the business environment of their software packages. The detailed technique to model business documents is best understood in the case study (cf. Table 2 and Fig. 7).

3.4 Business Service View

The fundamental principle of the business service view is to describe the business collaborations between network components. The business service workflow does not add any new information. Accordingly, the business service view artifacts are automatically created from the information gained in the previous workflows. For each role appearing in any business transaction a network component is created. In an application-to-application environment these components communicate with each other. Each business transaction is mapped to a service transaction defining the exchanges between the components (see Fig. 8). Exchanges are either business document exchanges or business signals sent as acknowledgment or failure notices.

Note that each business activity of a business transaction will result in an operation assigned to the corresponding network component. A service collaboration spans over all service transactions. A network component for a given role has to support all the operations defined in any business transaction (see Fig. 9 of the example). If a company is capable of a role in a business collaboration, it has to support all the operations of the corresponding network component and must keep the choreography defined in the collaboration. A company might support one business context, but not another. It follows that each network component must define the context drivers it supports in its profile.

4 The UMM Case Study

In this Section we present the UMM by means of two case studies. The first one is the order management of books and the second one is the order management of tourism products like hotels, flights etc. Due to space limitations we cannot describe the ex-

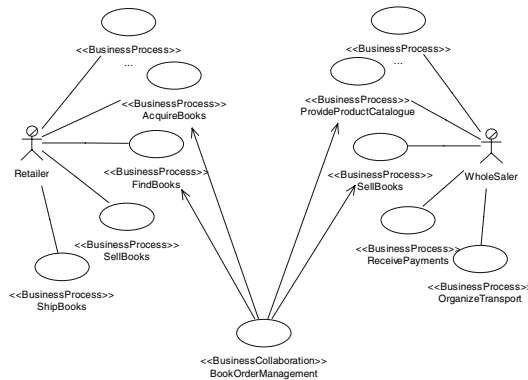


Fig. 1. Business Processes in the Business Domain

amples in all its real world requirements. Therefore we have chosen a limited functionality that still allows to understand the concepts of UMM. It is important to show that most of the functionality of the book example might apply to any other product. However, according to some context drivers like “product”, the basic scenario must be customized with respect to choreography and exchanged document types. If appropriate, we show the tourism counterpart of the book example as well as a corresponding generic example with context driver notations.

4.1 Business Domain View

In our example the stakeholders in the business domain of book selling are interviewed. Thus, book retailers and book wholesalers amongst others (which we do not concentrate on here) describe their business processes in the domain under consideration. Business processes are recorded using the UML concept of use cases. Fig. 1 depicts on the left side use cases that show some sample business processes important to the retailer and to the wholesaler. The details of the business processes manifested in the associated use case descriptions and optional activity diagrams are not presented due to space limitations.

4.2 Business Requirements View

Considering the business processes of our example as depicted in Fig. 1, a business collaboration for *book order management* between a retailer and a wholesaler is established. This business collaboration is aligned with the business processes *find books* and *acquire books* on the retailer’s side and *provide book catalog* and *sell books* on the wholesaler’s side. Thus, dependency relationships are created from the *book order management* collaboration at the bottom of Fig. 1 to these processes.

The stakeholders’ agreed shared view on the business collaboration book order management is documented in a corresponding use case description as depicted in Table 1.

According to the description of the business collaboration, the book order management involves the registration of the retailer, the search for books, the reservation of books, the presentation of reserved

Table 1. Book Order Management Description

Form: Business Collaboration	
Business Collaboration Name	Book Order Management
Description	A Retailer orders books from a wholesaler. Its up to the retailer to search books according certain criteria. The retailer gets back a list of corresponding books including price quotes. It is possible to put books on a reservation list at the wholesaler (books from this list can be canceled without charges). The retailer can order books either from the result list or from the reservation list. The retailer must be registered for both making purchase orders and reserving books.
Preconditions	none
Begins When	Either the retailer searches for books to be ordered OR wants to view his current status of the reservation list
Ends When	When the wholesaler accepts the order and a contract is established,
Exceptions	Retailer finally does not want to place an order; Wholesaler does not accept an order; Books are not available; Wholesaler does not register the customer
Postconditions	Order completed
Initiating Partner Type	Retailer
Responding/Receiving Partner Type	Wholesaler
Initiating Events	Retailer’s need for books
Terminating Events	Books are ordered
Scope	Search books in a catalog, manage books on a reservation list, customer registration process; establishing contracts for book orders;
Boundary	Managing purchase orders between retailer and wholesaler; it is limited to their communication and does not involve any communications with banks concerning payment or transporters concerning shipment;
Constraints	Business Collaboration is defined in the context of Books
Supporting Business Transactions and Business Collaborations	Register Retailer; Search Books; Reserve Books; Order Books; Present Reserved Books

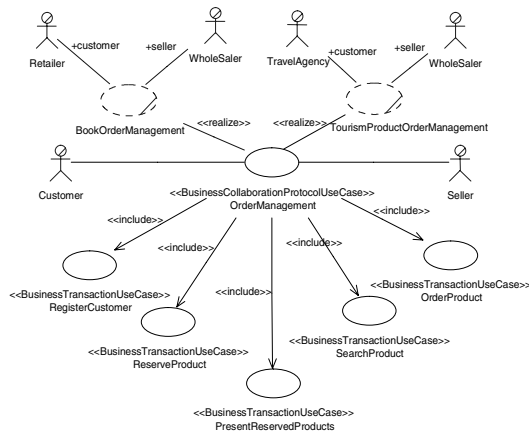


Fig. 2. Order Management Use Cases

books, and the ordering of books. These represent business collaboration use cases that are included in the book order management. Consider now an order management for tourism products. A use case description for tourism product order management is very similar to that for books. It differs only in the type of product and the initiating partner type that is a travel agency. The tourism product order management includes also very similar business collaboration use cases to that of book order management.

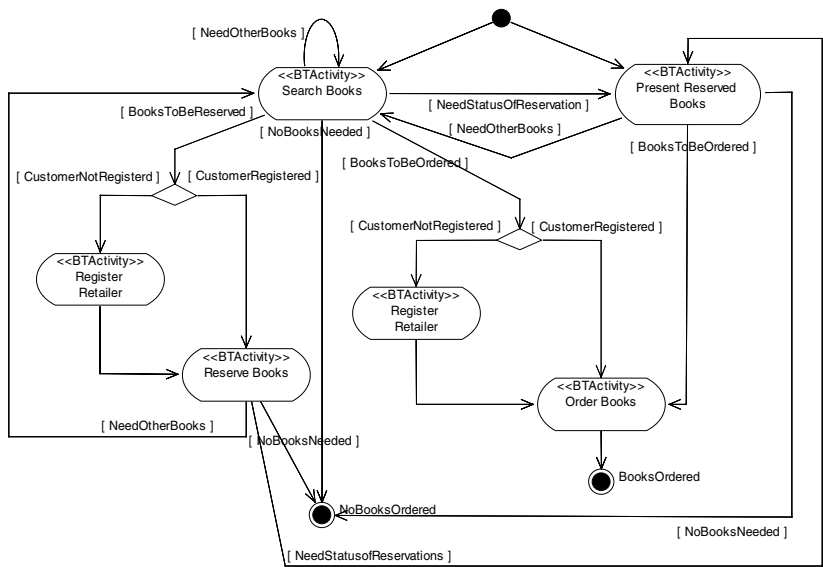


Fig. 3. Business Collaboration Protocol for Book Order Management

The similarity between book and tourism product order management mentioned above might be viewed as two different realizations of the same generic collaboration in two different contexts. This fact is presented in Fig. 2. We define a generic business collaboration use case *order management* which is performed by the actors *customer* and *seller*. It includes the business collaboration use cases *register retailer*, *search product*, *reserve product*, *present reserved products*, and *order products*. It should be noted that a business collaboration use case is defined in the UMM as the abstract generalization for business collaboration protocol use case and business transaction use case. We use the general term since at this point of the UMM process we do not know which of the two specializations applies. However, it will turn out later that all of the included use cases are business transaction use cases. Therefore, the corresponding stereotype is used in Fig. 2. In the *book order management* realization the *retailer* takes on the role of the *customer* and the *wholesaler* is in the role of the *seller*. Similarly, the *travel agency* takes on the role of the *customer* and the *wholesaler* is the *seller* in the *tourism product order management* realization.

4.3 Business Transaction View

The business collaboration protocol for our book order management example is depicted in Fig. 3. The choreography follows a description provided in the use case for the business collaboration realization. The book order management either begins by a search for books or by the query for the reservation list. After a search it is possible to order or reserve some books. Both activities require the retailer to be registered. If the result of a search was not satisfying another search is performed or the reserved books are queried. After a reservation of books was performed the next activity is either a new search or the query for the reserved books. Note that a query for books works for registered customers only, because otherwise they were not able to make a reservation. After querying the reserved books, some of these books might be ordered. The other choice is to perform a new search for books. The business collaboration always

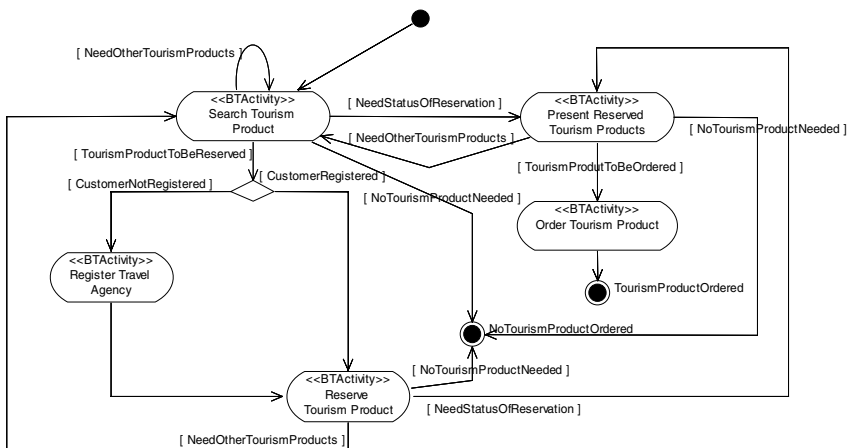


Fig. 4. Business Collaboration Protocol for Tourism Product Order Management

ends after ordering books. However, the search for books, the reservation, and the presentation of the reserved books might also be the last activity with the consequence that no books are ordered. Furthermore, we assume a maximum performance time of 24 hours for each business transaction activity and non of these being concurrent.

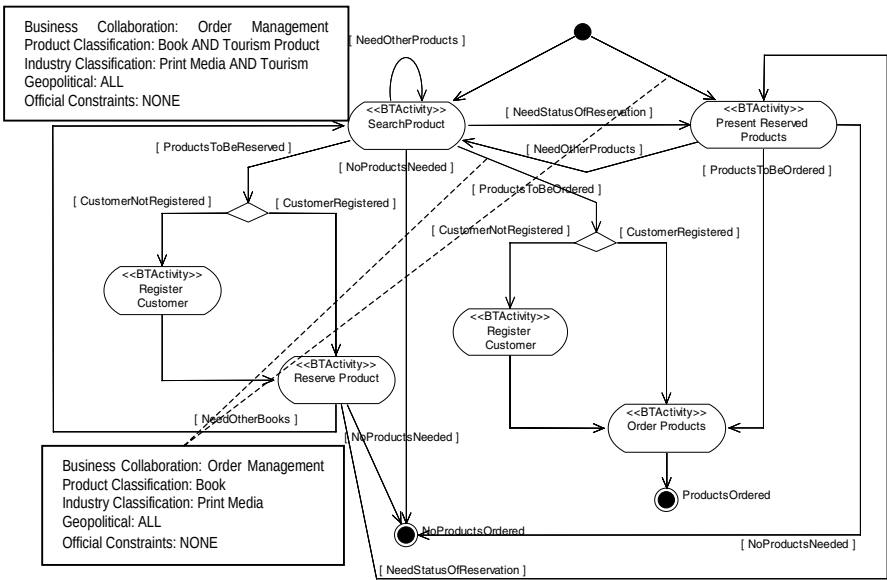


Fig. 5. Business Collaboration Protocol for “Generic” Order Management

In Fig. 4 we present the business collaboration protocol for our tourism product order management example. Again, the choreography must follow a detailed description captured in the business collaboration realization. In addition to the different type of product, we assume the following differences compared to the choreography of the book order management: The reservation list is only valid during the business collaboration, it does not survive the end of the business collaboration. Thus, it does not make sense to start the business collaboration with a query for the reserved tourism products. Furthermore, it is not possible to order the tourism products immediately after having performed the search. First, all products for a trip must be reserved. Orders include only products appearing on the reservation list.

Since the two business collaboration protocols are quite similar they share a common business collaboration protocol for our order management example. It must be noted that this fact was already discovered at the business requirements view, when we defined the book order management and tourism product order management to be realizations of the generic order management. Fig. 5 presents the common business collaboration protocol. When modeling with UMM, all the resulting models reflect the business environment the models are produced for. This business environment is characterized by different context drivers. Our example is modeled in the context of the order management business collaboration, books and tourism products as product types, and print media and tourism as industries. No other restrictions apply on the other context drivers. This fact is denoted by assigning the context drivers to the busi-

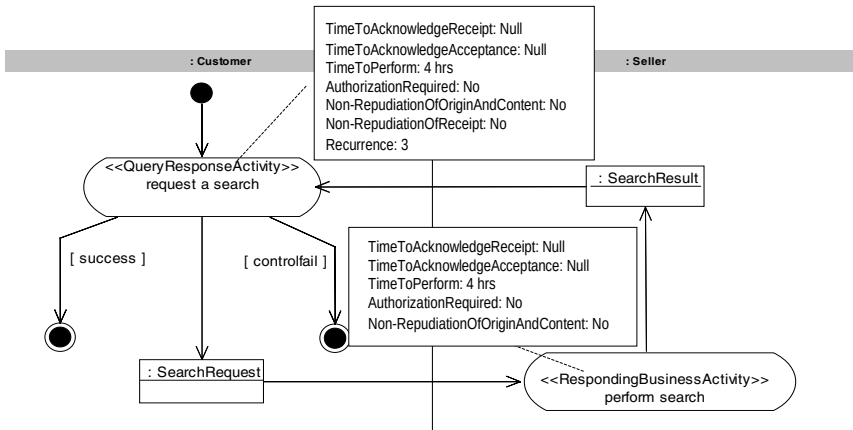


Fig. 6. Business Transaction “Search Product”

ness collaboration protocol. In Fig. 5 we present the context drivers in the upper left corner. However, according to the requirements the transitions from start state to present reserved books and from search product to order product is not valid in the tourism case. Thus, the transitions are not available in the default context, but in a context limited to books as products and print media as industry. The context drivers shown in the lower left corner of Fig. 5 are assigned to these transitions.

In the next step each of the business transaction activities is detailed by its own activity graph called business transaction. We present the business transaction search product in Fig. 6. As defined in the context drivers in the upper left corner of Fig. 6, the choreography for search product applies for both domains in our example. Since there is a response that does not immediately result in a contractual obligation and the responder has the information (about the products) already available, we selected the query/response pattern. The customer performs request a search as requesting activity, which is stereotyped according to the selected pattern. The responding activity perform search is executed by the seller. Since no obligations apply, no acknowledgments are needed and non-repudiation does not apply. Authorization is not required, because anyone is allowed to perform the search. We set the performance time of both the requesting and the responding business activity to 4 hours. Furthermore, the requesting business activity will retry to initiate the search 3 times in case of a control failure. The corresponding instantiation of the tagged values is denoted in Fig. 6 by comments assigned to the activities.

The requesting activity produces a *search request* document that is input to the responding activity. The responding activity outputs the *search result* document that is returned to the requesting activity. The structure of both document types is defined in class diagrams. In the following we concentrate on the *search result* document structure as an example. Table 2 presents a worksheet for the content description of the search result in our book example.

The worksheet to describe the document structure starts off with a general description of the document. It is the goal to map this description to a class diagram that is based on re-usable core components. Thus, the lower part of the worksheet is used to identify the core components used in the *search result* document. Unfortunately, there

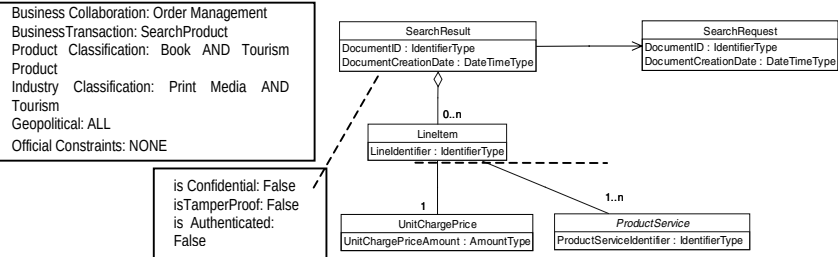
does not yet exist a globally agreed set of core components. Hence, the core components listed in Table 2 are still based on a pre-mature catalog developed for ebXML demonstration purposes in November 2001 [14]. Each row lists a basic core component. The representation term of a basic core component follows one out of 16 core component types. Core component types consist of a content component that carries the actual content plus supplementary components that give extra definition to the content. E.g., the ISBN number will go in the content component of an identifier type, whereas the fact that the identifier is of type ISBN goes into the supplementary one. Similarly, the quoted price itself instantiates the content component of an amount type, and the currency is denoted in a supplementary component. Furthermore, each basic core component is assigned to a so-called object-class. This gives a hint to which common business object (= class) the core component will be assigned to as an attribute. The business term provides a possible alternative term for the core component that is usually used by business people in the given context. The semantic description helps to further describe the core component. Given the document description in Table 2 it is easy to recognize that some core components are specific to a search result for books and others will appear in a result list for any product. This becomes even more evident by the corresponding description of the tourism product example which is not lined out due to space limitations. Given a significant overlay between the search result documents of our two examples, it is possible to create a class diagram for the common structures. This class diagram is depicted in Fig. 7a. The context drivers for the common search document are defined in the box left to the diagram. The tagged values in the comment assigned to the class search result denotes the fact that neither the search result document nor one of its components are confidential, tamper proof, or authorized. For reasons of simplicity, we define this flag only on the document level. Commonly, the search result is an aggregation of zero to more line items. Each line item refers to one or more product/service and exactly one unit charge price. The types of the attributes correspond to core component types. The search result class covers the unique document id assigned by the seller and the document creation date. The line identifier corresponds to the sequence number of the line item in the result list. The unit charge price amount attribute of class unit charge price is self explanatory.

The structure of each product/service in the result list depends on the context of the product, and must hence be replaced by a specialized item in a given context. It follows that the class product/service in the common class diagram is abstract. A product/service identifier is common to all specialized items and is thus an attribute of the generalized class. In Fig. 7b we present the search result structure for the book example. Given the context drivers on the left, the abstract product/service class is replaced by a specialized book item. Book item inherits the product/service identifier and the product/service description text for its title. Furthermore it defines author as an additional attribute. Fig. 7c shows the search result structure in the tourism domain. In this case specializing the product/service as a tourism item is still too general. Since the structure of a tourism item varies from one tourism product type to another, a further specialization for different tourism product types is required. In the example depicted in Fig. 7c the context drivers for flight item, hotel item, etc., follow the product type.

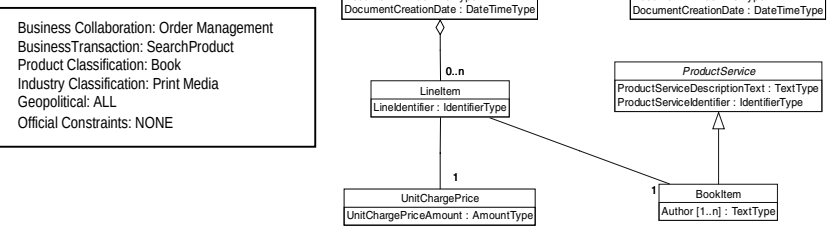
Table 2. Document Description for the Search Result Document

Form: Content Description				
Content Description Name	SearchResult			
Content Description	A search result document is identified by a unique id assigned by the seller. Additionally, the search result document includes its creation date and time. The search result document returns a list of numbered items (books) that meet the search criteria. If no book matches the search criteria the list will be empty. Each item on the returned list refers to exactly one book. Furthermore, a price quote is made for each item. A book is described by the ISBN as unique identification, a title and its author(s).			
Attribute	Representation Term	Object Class	Business Term	Semantic Description
DocumentID	IdentifierType	SearchResult	Search Result ID	Unique id of the search result doc.
DocumentCreationDate	DateTimeType	SearchResult	Search Result Date	Creation date of the search result doc.
LineIdentifier	IdentifierType	LineItem	Line Number	Sequence number of the line item
ProductServiceDescriptionText	TextType	ProductService	Title	Title of the book
ProductServiceIdentifier	IdentifierType	ProductService	ISBN	ISBN of the book
Author	TextType	BookItem	Author	Author(s) of the book
UnitChargePriceAmount	AmountType	UnitChargePrice	Price Quote	Price quote for the corresponding book

(a)



(b)



(c)

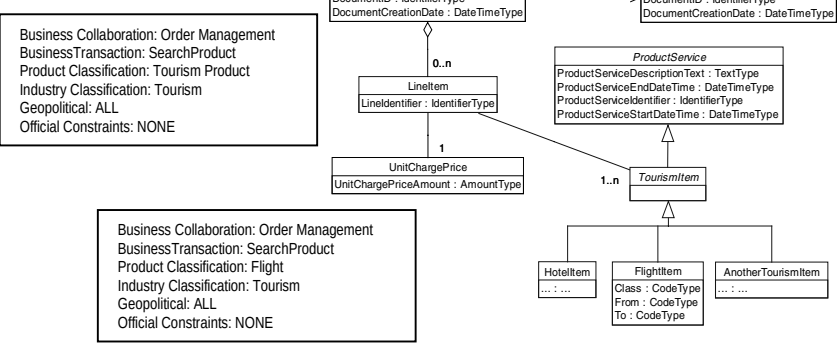


Fig. 7. Search Result Document: (a) Common, (b) Book, (c) Tourism

4.4 Business Service View

During the business service view the message exchanges between the network components are defined. In our example there exist two network components: one for the customer service and one for the seller service. Each business transaction of the BTV maps to a corresponding service transaction in the BSV. Fig. 8 depicts the search product service transaction. The corresponding business transaction does not require any acknowledgments. Therefore, the resulting sequence diagram is quite simple and only includes document exchanges. The customer service calls the performSearch operation of the seller service by sending the search request document. The seller service returns the search result document by calling the requestASearch operation that initiated the service transaction. By mapping all five business transactions of our example to service transactions, we exactly define the services to be supported by each network component. Fig. 9 shows the resulting network components and their services. Network components are represented as interface classes and their services as operations. Software vendors supporting a role in the simple order management must implement the corresponding interface class in their applications. The unambiguous input to a service was defined in the class diagram of the BTV. The collaboration protocol (Fig. 5) defines the choreography among transactions, whereas the sequence diagram of the BSV (Fig. 8) defines the choreography within a transaction.

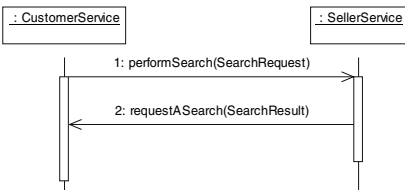


Fig. 8. Service Transaction Search Product

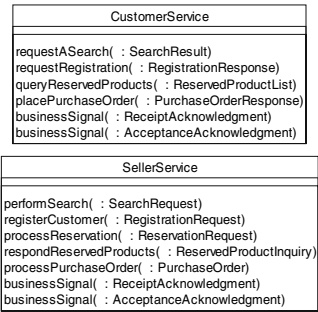


Fig. 9. Network Components

5 Summary

In this paper we present UN/CEFACT’s modeling methodology (UMM). The goal of UMM is to capture the business knowledge that enables the development of low cost software components by software vendors to help the small- and medium-size companies. The software process-like methodology is based on 4 workflows: (1) The business domain workflow elicits and organizes business processes and information in the business-to-business domain. This includes the discovery of existing business processes of importance to stakeholders and their categorization. (2) The business requirements workflow uses the discovered business process to identify possible collaborations. Furthermore, the workflow includes the creation of detailed requirements for the business collaboration documented in UMM use case diagrams. (3) The business transaction workflow further elaborates the requirement use cases by detailing the activities that occur and defining their choreography in activity diagrams. Furthermore, the structure of business documents exchanged in business transactions is

modeled in class diagrams. (4) The business service workflow describes the business collaborations amongst network components. The network components are described as interface classes that are integrated by software vendors into their products. The workflow precisely defines the dynamics involved and the operations necessary to support a role in the collaboration.

All the workflows are detailed by means of examples on ordering/selling books and tourism products. In these collaborations the choreography as well as the document structures are very similar, but differ in some details. In order to avoid a proliferation of similar business collaborations, a method unambiguously adapting the common collaboration to the specific business environment is needed. Thus, we have extended the current version of UMM by the concept of context drivers. This allows software reuse for the common parts of a collaboration and unambiguous, context-specific adaptations. Of course, it is recognized that even with UMM, the issue of businesses doing things differently would not disappear. However, in regard to the SMEs, it is envisioned that software providers would create applications that implement the most popular scenarios.

The example introduced in this paper will serve as reference example for our future work items. Topics of interest include a more formal definition of the context drivers. Appropriate code lists, e.g. UN/SPSC for products and services, must be identified to ensure clear semantic definitions and machine-processable context drivers. Furthermore the context drivers assigned to UMM stereotypes must be machine processable. The object constraint language (OCL) seems to be a candidate for this purpose. Additionally, we are working on the transformation of the UMM artifacts into current B2B technologies, such as ebXML [3,7] and Web Services [20].

References

1. Booch, G., Jacobson, I., Rumbaugh J.: The Unified Modeling Language User Guide. Addison Wesley Object Technology Series, Reading, (1998)
2. Dogac, A., Tambag, Y., Pembecioglu, P., Pektas, S., Laleci, G. B., Kurt, G., Toprak, S., Kabak, Y.: An ebXML Infrastructure Implementation through UDDI Registries and RosettaNet PIPs. ACM SIGMOD International Conference on Management of Data, (2002)
3. ebXML: Homepage of the ebXML Initiative. <http://www.ebXML.org>
4. ebXML: ebXML Technical Architecture Specification v1.04, (2001) <http://www.ebxml.org/specs/ebTA.pdf>
5. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. <http://informatik.uibk.ac.at/users/c70385/wese/wsmf.paper.pdf>
6. Hill, N.C., Ferguson, D.M.: Electronic Data Interchange: A Definition and Perspective. EDI Forum: The Journal of Electronic Data Interchange, Vol. 1, Issue 1, pp. 5-12, (1989)
7. Hofreiter, B., Huemer, C., Klas, W.: ebXML: Status, Research Issues and Obstacles, Proc. of 12th Int. Workshop on Research Issues on Data Engineering (RIDE02), San Jose (2002)
8. Huemer, C.: Defining Electronic Data Interchange Transactions with UML. Proceedings of HICSS-34, Maui, (2001)
9. Huemer, C.: <<DIR>>-XML² -Unambiguous Access to XML-based Business Documents in B2B E-Commerce. Proc. of 3rd ACM Conference on Electronic Commerce, Tampa (2001)
10. ISO: Open-edi Reference Model. ISO/IEC JTC 1/SC30 ISO Standard 14662 (1995)

11. Lee, R.M.: Documentary Petri Nets: A Modeling Representation for Electronic Trade Procedures. Business Process Managements - models, techniques and empirical studies. Springer LNCS Vol. 1806 (2000)
12. Lenz, K., Oberweis, A.: Interorganizational Business Process Management with XML Nets. Advances in Petri Nets. Springer LNCS Vol. 2472 (2003)
13. Ling, S., Loke S.W.: Advanced Petri Nets for Modelling Mobile Agent Enabled Interorganizational Workflows. Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2002), Lund (2002)
14. UN/CEFACT TMG: Catalog of Core Components. Version November 2001.
http://webster.disa.org/cefact-groups/tmg/downloads/CCWG/drafts/CC-Catalogue_11_01.zip
15. UN/CEFACT TMG: UN/CEFACT Modelling Methodology.
<http://webster.disa.org/cefact-groups/tmg>
16. UN/CEFACT TMG: UMM Meta Model. Revision 12.
<http://webster.disa.org/cefact-groups/tmg/downloads/BPWG/drafts/UMM-MM-V20030117.zip>
17. UN/CEFACT TMG, Core Component Technical Specification, Version 1.90, (2002)
http://webster.disa.org/cefact-groups/tmg/downloads/CCWG/for_review/CCTS_V_1pt90.zip
18. van der Aalst, W.M.P.: Modeling and Analyzing Interorganizational Workflows. Proc. of Int. Conf. on Application of Concurrency to System Design (CSD'98), Fukushima (1998)
19. van der Aalst, W.M.P.: Process-Oriented Architectures For Electronic Commerce and Interorganizational Workflow. Information Systems, Vol. 24, No. 8, Elsevier Sciences Ltd (2000)
20. W3C: Web Services Activity. <http://www.w3.org/2002/ws/>

Security Framework for Mobile Agent Platforms (SFMAP)

Terry Kapnoullas¹, Elizabeth Chang², Tharam Dillon³, and Ernesto Damiani⁴

¹ Department of Computer Science and Computer Engineering, La Trobe University, Bundoora
Victoria, Australia 3083
terry@cs.latrobe.edu.au

² School of Information Systems, Curtin University, Western Australia, Australia 6845
e.chang@tpg.com.au

³ Faculty of Information Technology, University of Technology, Broadway, New South Wales
Australia 2007
tharem5@it.uts.edu.au

⁴ Department of Computer Technology, University of Milan, Crema, Italy
edamiani@crema.unimi.it

Abstract. Security Framework for Mobile Agent Platforms or SFMAP key feature of this approach is to provide for multiple levels of access control for agents that arrive at a server, depending on their level of trust. This level is dependent on the previous history of servers visited. This necessitates an entirely new approach to authentication and authorisation based on history plus identity, rather than just identity. The designed framework contains many different services for which permissions can be defined, including system and database resources.

1 Introduction

In its early days the World Wide Web and Internet, allowed a computer to essentially act as a gateway into an electronic world of information. Currently we are using an increasing number of web applications that provide different web services. A result of this is that more and more computation will occur on the Internet rather than just make use of it as a gateway. Programs will be distributed throughout a network and the World Wide Web and work together in a loosely integrated fashion.

Because the Internet is a best-effort network, it is common for a user to suffer from long waiting times when searching and comparing data on the Internet. An approach that has been developed to address this, and other issues, is mobile agents. Although mobile agents have been around for some years now; they are still in their infancy, as they still have security issues that need to be addressed, to enable them to be used in a wide variety of applications. The main focus of this paper is to focus on the security issues surrounding mobile agents. In particular formulating a method to protect mobile agent platforms from malicious mobile agents.

2 Mobile Agent Security Issues

Security attacks on mobile agents can commonly be categorized into four inclusive classes, these being corruption of information, denial of service, disclosure of information, and interference or nuisance. It is essential to observe that several of the security issues that are evaluated in this paper have equivalents in classical client-server systems. Mobile agents, unfortunately, allow new methods for misuse and exploitation, extensively increasing the level of security attacks. Generally the new security issues that are due to mobile agents arise from the different aspects of their architecture, rather than the historical client/server ones. The security areas of mobile agents can be categorised by two key components, the mobile agent and the mobile agent platform. By analysing these two components, four threat categories can be identified:

- mobile agent attacks execution platform
- mobile agent attacks mobile agent
- execution platform attacks mobile agent
- other attacks execution platform

3 Security Investigation Key Concepts

Various concepts have been researched and designed to develop the security framework for mobile agent platforms (SFMAP). These concepts are in accordance with preventing security threats to a mobile agent platform, as well as the agents executing on it. In addition, the techniques used are illustrated in the platform security framework for mobile agent platforms to ensure its security while still allowing it to be accommodating to various access permission levels.

3.1 Allocating Mobile Agent Access Rights

The method that the host platform uses to allocate mobile agent access rights is to use a combination of path histories, SPKI certificates and an elaboration of the Java sandbox model.

The path histories security countermeasure stores all of the host platforms that a mobile agent has visited. The host platform that the mobile agent is currently at can then view the visited host platforms and decide whether to process the mobile agent, and if so how to process it. This security measure requires each of the host platforms that a mobile agent visits to sign it with its unique key and specify the next platform that the mobile agent is to visit, to supply the next platform with the mobile agent's entire history.

To avoid manipulation of a mobile agents, the signature of the new path entry must include the prior entry in the execution of the message digest. When the mobile agent arrives at the next host platform, the platform can work out if it trusts the host platform this particular mobile agent has visited, by examining the list of identities sup-

plied by individually authenticating the signatures of each entry in the path history to confirm identity.

Although this security countermeasure does not stop a mobile agent platform from maliciously attacking a mobile agent, it can warn other host platforms of potential danger and act as a deterrent, since the platform signature on the agent proves that it committed the act, damaging the host platforms reputation. However, there is a disadvantage to this security measure as the mobile agent travels to multiple sites, the path history increases. Also this security measure also relies on the ability of a host platform to judge whether or not it trusts the sites a mobile agent has visited.⁴⁵

3.2 Multilevel Sandbox Model

The Java security model has three methods of defense from attacks of malicious code: The Byte Code Verifier, the Class Loader, and the Security Manager. These security measures together carry out load and run-time checks to control access to system resources. All the security measures tightly interact for increased security, however because they strongly depend on each others security measure all could fail if there is a single anomaly in any of them.

Currently with the Sandbox Model, code is either trusted or is not trusted. However for this security framework to work it is necessary for mobile agents to be assigned access levels and to dynamically allocate the host platforms system resources or services.

Due to the Java byte code verifier, class loader and security manager being tightly integrated, the research on Java has shown that by creating a security manager to directly modify the Java Virtual Machine permissions would allow security permissions for system resources to be allocated dynamically. This is because any applications invoked with a security manager must be granted explicit permission to access local system resources apart from read access to the directory and its subdirectories where the program is invoked. The Java platform provides permissions to allow various levels of access to different types of local information.

Because permissions let the mobile agent platform override the default security policy, much caution must be taken when assigning permissions so as not to create an opening for malicious code that may attack the host system.

Permissions are granted to a program with a Java policy file. A policy file contains permissions for specific access when using the Java virtual machine. Permission consists of the permission name, a target, and in some cases, a list of actions. One permission file is created for each access level and is loaded dynamically for the corresponding validated mobile agent.

By using this method, the security framework is capable of assigning levels of trust or system permissions to a mobile agent by using the multilevel sandbox model.

Therefore no longer is a mobile agent trusted or not trusted by the security framework, but is assigned a level of trust, where there is no limit of available levels¹¹.

3.3 Methods of Assigning Permissions

When a mobile agent arrives at a host platform it is put in quarantine within SFMAP. This essentially means that the mobile agent remains stateless as it is waiting to be

validated and granted access to system resources to execute. SFMAP retrieves all of the security certificates from the mobile agent for analysis while the mobile agent is in this state.

Once the security framework has obtained the certificates it will then read through all the information in the SPKI certificate, checking the level of trust for each platform an agent has visited. Once this is completed the mobile agent will be assigned the access level of the least trusted platform that it has visited.

However, a method of assigning permissions based on identification needs to be addressed in order to achieve what is previously described in SFMAP. Two methods of assigning permissions based on identification were researched in consideration for the mobile agent platform security framework. The two methods are a web of trust, and absolute authority, which are described in this section of this paper.

In a web of trust, personal acquaintances sign each other's keys to establish identity within their own ad-hoc community. In a mobile agent relationship, when a mobile agent arrives at a platform for the first time, a level of trust for that mobile agent needs to be determined and that the provided signature key is the claimed mobile agent platform. The mobile agent's policy is then used to search through a list of subsequently trusted mobile agent platform keys to assemble a continuous path. In other words, an already trusted agent platform trusts the new agent platform. If this path assembly occurs, the agent platform will assign the platform a security level and store its key in the platforms collection.

However the methods of assigning access levels to a mobile agent platform have many variations, but typically use the following idea. The mobile agent will only be as trusted as much as the platform that recommends it. However the larger the path that is constructed the less a mobile agent is trusted, thus after a certain point, the agents would not really be trusted at all.

This method, although applicable to a mobile agent architecture, is not really appropriate as the security framework that is being designed desires specific control over each mobile agent platform visited. Due to its nature, the platform relies on a great deal of trust and because of this nature; it still has vulnerabilities if another trusted platform has any vulnerabilities.

Absolute authority relies on a mobile agent platform already obtaining a list of all keys from the mobile agent platforms it interacts with. When a mobile agent arrives at the host platform, it presents its key. That key is then compared to the keys that the platform already possesses. If the mobile agent is correctly validated it is assigned a permission already assigned by the host platform, otherwise if the key is invalid or not present, the mobile agent is denied access to the host platforms services or only given limited access.

Absolute authority was chosen for the mobile agent security framework, as it provides a greater degree of control of the visiting mobile agent compared to the web of trust method. However using the web of trust to assign host platform permissions may be incorporated in a later version of SFMAP.

3.4 Path Histories

The path histories security countermeasure is to store all of the host platforms that a mobile agent has visited. The host platform that the mobile agent is currently at can then view the visited host platforms via SPKI Certificates and decide which access

level to grant it. This security framework requires each of the host platforms that a mobile agent visits to sign it with its unique key and specify the next platform that the mobile agent is to visit. In this way supplying the next platform with the mobile agents' entire history. The security framework then checks that the site that the agent visited actually did add its own SPKI certificate. If the next target platform in the SPKI certificate does not match the one actually added, the platform will conclude that this is a higher risk mobile agent and assign it a lower access level to the system resources.

The SPKI certificates are stored in a vector within the mobile agent that can be only accessed by the mobile agent platform. A platform has been given access to protected methods that can add and read certificates but cannot modify or delete any.

This will allow mobile agent platforms to access and analyse mobile agents SPKI certificates and then grant the mobile agent the allocated resources it is allowed.

3.5 SPKI Certificates

The Java Development Kit (JDK) has incorporated the design and implementation of cryptographic keys, signatures and certificates since version 1.0, and has been continually enhanced with every new release. However, the certificate interfaces still only cover identity certificates. As additional security applications and tool make use of authorisation certificates, the model of an authorisation certificate and its implementation need to be added to the Java Security API ⁷.

In conventional qualification based classification, the means by which this is accomplished is by a secret ticket that permits permission to anything presenting it, and so the qualifications are required to be rigorously restricted. The SPKI certificate, however, only permits authority to the key specified in the certificate. Consequently, it does not need to be treated as secret and have to be strictly controlled.

The main purpose of the SPKI certificates is to establish authorization in a fairly ad-hoc, "web of trust" like way. SPKI has name certificates, but the names use local name spaces and are only for the convenience of the users, the identities are represented directly by cryptographic keys. Due to the nature of SPKI certificates, they are better suited for authentication of mobile agents compared to traditional certificates. This is because mobile agents' nature is to visit many sites making it hard to control who receives the certificates. Using SPKI certificates, this is no longer an issue. ^{7 8 9}

4 Certificate XML Design

Certificate design is extremely important, as it will be the main method to validate where a mobile agent has been. Presented here is a XML Schema for the encoding and validation of SPKI certificate. The XML Schema is based on the [IETF Draft] "SPKI Certificate Structure". However this needs to be extended to allow for history. The main emphasis of this document is on the encoding of all SPKI constructs under XML and ensuring that it carries all the relevant information that can be validated and show the path history of a mobile agent.

4.1 Certificate XML Scheme

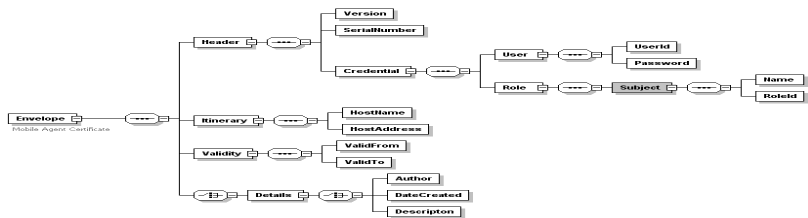


Fig. 1. Diagram view of SPKI certificate shows the XML Schema for a mobile agent certificate. It demonstrates the structure and relationships of the security certificate.

5 Platform XML Design

Being able to configure the platform system recourse permissions is a major part of this paper, as it will be the main method used to allocate how much a mobile agent can do on the system based on its level of trust. Described in this section are the XML schema and an example XML file of how this is used.

5.1 Platform XML Scheme

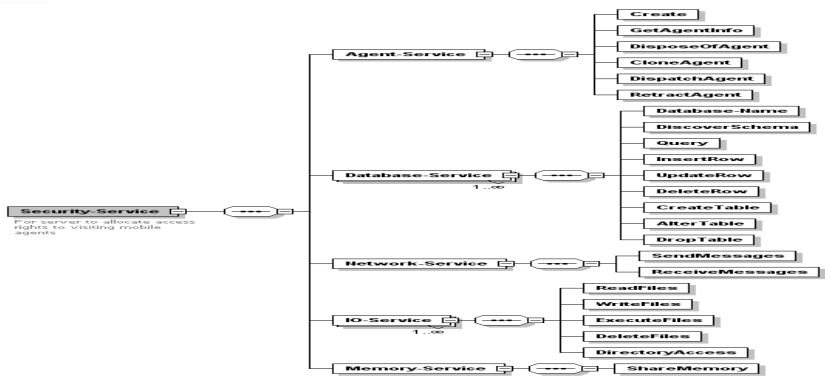


Fig. 2. Diagram View of Platform XML Scheme. This diagram shows how the entire mobile agent services branch off the main security service, and from those major services, the sub services that the system caters for. The services have been grouped into their relevant relationships for easy accessibility and maintenance to the security permissions for a service.

6 Architecture of the Mobile Agent Platform

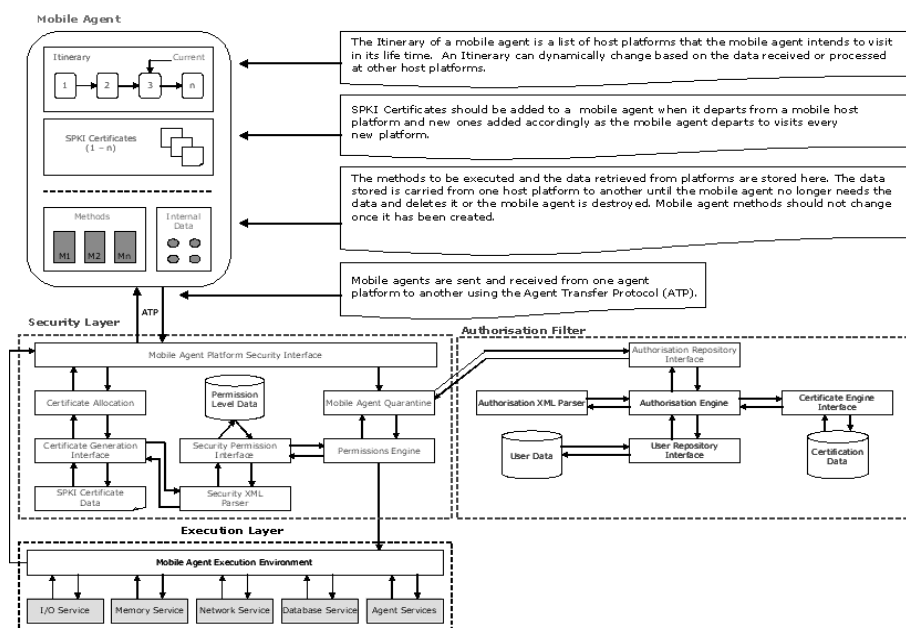
The architecture of the mobile agent platform has been modified from the standard platform to incorporate the security framework. A standard mobile agent platform

generally consists of a mobile agent, an execution environment and system services. The new security framework adds to this a security layer for setting a mobile agents' access level and permissions, as well as an authorization filter for validating the certificates associated with the previous hosts that a mobile agent has visited. Note in this case that we are authenticating based on identity as well as history. Therefore the sequences of identity authentications is,

$$\text{authentication}_1, \text{authentication}_2, \text{authentication}_3, \dots, \text{authentication}_n \quad (1)$$

6.1 Architecture Diagram

This architecture is shown in figure 4.0 on the following page and a description of all of the components is also described later in this paper.



6.2 Security Layer

In this section, we will briefly outline the functionality related to each of the components in the security layer. This together with the authorisation filter is the essence of the security framework.

Mobile Agent Quarantine component is used to hold a mobile agent that has just arrived, preventing it from executing any code. The mobile agent quarantine then sends the mobile agents' certificates to the authorisation filter to be validated and find out its level of trust. Once this is obtained the component sends the level of trust to the permissions engine.

Permission Engine is the main component of the security layer. Its role is to set the system resources that a mobile agent has access to, based on the mobile agents' level of trust and prevent it from having access to any unauthorized system resources once it is executing.

XML Parser is the component responsible for conversion from the textual representation of the data into a format that is accessible by the application. The XML parser design is based on the DOM, which is a W3C standard. The reason for using the DOM representation is that authorisation objects are defined via a generic expression of the XPath language, and the evaluation of these expressions may require one to navigate the XML structure in arbitrary ways.⁸

Permission Level Data is an XML file containing the permissions granted to each level of trust for a mobile agent.

Certificate Allocation component is used to assign a SPKI certificate to a mobile agent when it departs a mobile agent platform. It uses the SPKI certificate data as a template and fills in the required information that is required for the mobile agents journey.

SPKI Certificate Data an XML template that contains the basic form of the security certificate and all of its required fields.

6.3 Authorisation Filter

The purpose of the authorisation filter is to validate any keys that a mobile agent may present to the host platform as being valid or not. It does this by checking the key with a repository of already collected keys, and compares them to see if the key belongs to the corresponding mobile agent. The authorisation filter is located next to the security layer as that is where it will receive all of its keys to validate against.

User Repository. The user repository maintains a description of all the mobile agent platforms and groups. The user repository is stored in a database and has an interface used to access all of the data contained within. It will contain information such as user identification, hashed passwords, level of trust, etc.⁸

Authorisation Repository. The authorisation repository maintains the collection of authorisations that describe the security policy. The services offered by the authentication repository, permit the retrieval of the authorisations applicable to a given request.⁸

Certificate Engine. The certificate engine is used to validate certificates that a subject may present to substantiate a mobile agents ability for a certain role.⁸

Authorisation Engine. The authorisation engine is the main component of the authorisation filter and coordinates the rest of its components. It reacts to the SPKI XML certificate and parses the header element to determine its subject. If the subject presents a username and password, the engine then validates the usernames and passwords, and sends the results back to the component that called it.⁸

XML Parser. The XML parser is the component in the authorisation filter is the same as the one in the security layer previously described.

6.4 Restricted Resources

The resources listed in this section of the document are restricted, as they could possibly be misused by a mobile agent for malicious purposes.

I/O Resources being Input/Output (I/O) facilities are based on streams, which provide simple ways to read and write data of different types. This is very important as it is quite likely that mobile agents will need access to files on the host platform; however the host platform may want to specify who can use which services.

Added to the design of this framework under I/O resources is also the ability for a mobile agent to delete files on the system, or to even execute another program on the host platform.

Read File service, mobile agents can open a file of any type and extract streams of data from within it. This allows a mobile agent, for example, to read an XML file of the host platforms list for which a mobile agent can visit, save this data and add the sites to its itinerary.

Write Files service, mobile agents can open or create files on the host platform and insert streams of data prepared previously by the mobile agent into a file of any type. This allows a mobile agent, for example, to write or append to an XML file of the host platforms that it recommends other mobile agents to visit. Thus it does not need to stay on the host platform and message them itself.

However if the file is set by the operating system to be read only, an I/O exception will occur preventing any modification to the file.

Execute Files service is used to allow mobile agents to make system calls to execute applications that are external to the host platform and that cannot be accessed from within it.

Delete Files service, mobile agents can remove files from the host platform. This allows a mobile agent, for example, to delete any temporary storage files that it now deems deprecated, such as an old XML file of host platforms that it would like updated with a new one. However if the file is set by the operating system to be read only, an I/O exception will occur preventing any modification to the file.

Directory Access is not exactly an I/O Resource, but rather an element that can be used to specify a specific directory or directories on the host system where a mobile agent can use a particular I/O service. Thus, every time an I/O resource is allocated, a directory can also be allocated for it, thus limiting access that a mobile agent has to a host platform.

For instance the host platform could specify that mobile agents from a particular host can read all files in the directory 'C:\ReadMe' on the server and write files to the directory 'C:\WriteHere'. The system also allows the administrator to the host platform to specify an entire directory or root directory and its sub-directories, such as 'C:\'. This would allow the allocated mobile agents access to the entire drive C. However this is not recommended as you would be allowing the mobile agent total access to the host system for that particular I/O resource.

Memory Resources is important as when two mobile agents communicate or merely co-exist in one server, they should be protected from tampering with each other. Therefore the architecture of the host platform needs to provide isolated memory domains to prevent an agent spying on another agent or its processing data, or even corrupting the memory with garbage or missing information to prevent it from acquiring the desired results.

Shared Memory is important to isolate one mobile agents' memory preventing access from other agents. There may be exceptions however, especially with parallel processing where agents being able to access each others' memory could prove to be a major performance benefit. This is why the setting in this framework has the ability to be over written if need be.

However, although this has been incorporated into the design, it does not need to be developed due to the nature of Java. When the Java virtual machine executes any code it automatically places it into its own memory space preventing any other program executing on the same virtual machine from tampering with each other. The disadvantage to this however, is that if a developer of a mobile agent system wanted a mobile agent to access the same memory, they will not be able to do so.

Network Resources Computer network communication makes use of the data processing and storage capabilities of the computer and the data transmission capability to relay messages from the user or program, to another. It resembles the regular postal service in its method of message routing and delivery, and fax transmission in its use of the telephone to send written messages.

Send Messages resources allow mobile agents to sent network messages to other mobile agents executing on another host platform or the host platform itself.

Receive Messages resources allow mobile agents to receive network messages from other mobile agents executing on another host platform or host platforms of the host platform itself.

Database Resources are the resources that a mobile agent platform provides for a mobile agent to use but only if the platform has a database for them to access. The services listed below are common for many mobile databases and can be given access levels by the security framework on whether or not a mobile agent has the right to use them.

Database-Name, On a platform there may be more than one database residing on it. The database name resource will allow a mobile agent to specify the name of the database it wishes to query. However this can only be done to specific database it has access rights to.

Discover Schema resource, allows a mobile agent to ascertain what tables reside on the database as well as the attributes and their types that belong to the table.

Query, A set of conditions created for the purpose of loading specific records from the selected database into the selected set.

Insert Row, A mobile agent can use the insert row resource to add records to a table within a specified database.

Update Row resource is used to edit values for attributes in one or more records of a relational table within a specified database.

Delete Row resource is used to delete values for attributes in one or more records of a relational table within a specified database.

Create Table resource is used to add a table and all of its attributes to a specified database.

Alter Table resource is used to modify a table and all of its attributes from within a specified database.

Drop Table resource is used to remove a table and all of its attributes permanently from within a specified database.

Agent Resources are the resources that a mobile agent platform provides for a mobile agent to use. The services listed below are common for many mobile agent

platforms and can be given access levels by the security framework on whether or not a mobile agent has the right to use them.

Create resource allows one mobile agent to generate a brand new mobile agent. Its state is initialised, and it starts executing.

Clone resource allows cloning a twin mobile agent. The current state of the original is duplicated into the clone.

Dispatched resource transmits a mobile agent travels to a new host, and its current state travels with it.

Retracted resource withdraws a mobile agent previously dispatched is brought back from a remote host and its state comes back with it.

Deactivated resource disables a mobile agent is goes to sleep. Its state as well as its data is stored on a disk somewhere.

Activated resource turns on a deactivated mobile agent is brought back to life as its state is restored from a disk.

Destroyed resource deletes a mobile agent. It as well as the information it stores is lost forever.

7 Results

In this investigation the effects of the mobile agent security framework in protecting a host platform from a malicious mobile agent are tested. The testing procedure is described in some detail, as well as what exactly was tested to validate that this security framework was a success.

7.1 Experiments with Validating Mobile Agents

Described in this section are the basic overview methods used in how the mobile agent platform security framework is validated. In each case the experiments were carried out using a single or multiple host validation. These experiments were carried out using personal computers over a local area network at La Trobe University computer science laboratories.

7.2 Validating Using a Single Host

When validating with a single host, only two computers were used. These experiments involved mobile agents being created on Agent Platform A and being sent across the network using the agents transfer protocol (ATP) to Agent Platform C.

Once the mobile agent is at host platform two it is validated and permissions are set for that mobile agent by the security framework, so it can only access the system resources it is allocated.

7.3 Validating Using Multiple Hosts

When validating with multiple hosts, three computers were used. These experiments involved mobile agents being created on Agent Platform A and being sent across the

network using the agents transfer protocol (ATP) to Agent Platform B. Agent platforms A and B had a varying level of trust. The mobile agent then is sent to Agent Platform C, using once again ATP.

Once the mobile agent is at the host platform C, it is validated based on the levels of trust from its path history and permissions are set for that mobile agent by the security framework. Therefore it can only access the system resources it is allocated.

7.4 Explanation of Results

The results of the experiments carried out in these mobile agent platforms are explained in detail so they can be clearly understood. All results are displayed in table format, which will now be described below.

Experiment Type column describes if the experiment was carried out on a singles host, or multiple hosts as described previously.

Platform A Allowed column describes whether or not platform A in the experiments, has access to the tested system resource or not.

Platform B Allowed column describes whether or not platform B in the experiments, has access to the tested system resource or not. If the experiment type is single host, platform B is not used and displayed as N/A.

Expected Results column describes what the predicted outcome of the test case will be. These are the desired results and should match the actual results column.

Actual Results column describes what the genuine outcome of the test case was. These results demonstrate if the experiment was a success if the actual results matches the expected results.

7.5 Experiments with Different Access Rights

Read Access service, mobile agents can open a file of any type and extract streams of data from within it.

Write Access allows a mobile agent to open or create files on the host platform and insert streams of data prepared previously by the mobile agent into a file of any type.

Execute Access service is used to allow mobile agents to make system calls to execute applications that are external to the host platform and that cannot be accessed from within it.

Table 1. All of the outcomes for the experiments mentioned previously are shown in the following table of results

Experiment Type	Platform A Allowed	Platform B Allowed	Expected Result	Actual Result
Single Host	Allowed	N/A	Enabled	Enabled
Single Host	Disallowed	N/A	Disabled	Disabled
Multiple Hosts	Allowed	Allowed	Enabled	Enabled
Multiple Hosts	Allowed	Disallowed	Disabled	Disabled
Multiple Hosts	Disallowed	Allowed	Disabled	Disabled
Multiple Hosts	Disallowed	Disallowed	Disabled	Disabled

Receiving Messages via a network involves a mobile agent executing on the host platform having the capability to obtain network messages. These messages could be from another mobile agent, host platform or another application.

Sending Messages via a network involves a mobile agent executing on the host platform having the capability to transmit network messages. These messages could be to another mobile agent, host platform or another application.

7.6 Summary of Results

The current mobile agent platform security framework was a success, as it demonstrates how it uses the mobile agents' path histories to validate an agent, and can calculate its level of trust. From this level of trust it successfully sets the mobile agents permissions to various system resources to prevent it from obtaining access to any that it is not allowed, without preventing it from accessing resources it is allowed to access. This lowers the risk of platforms that are not trusted sending mobile agents to run malicious code on another host platform.

However the mobile agent platform security framework in theory, has the ability to control many more system resources than tested. In theory the security framework should be capable of handling any permission to system resources that the Java Virtual Machine has control over.

8 Conclusion

In the near future computer processes will gradually be handled differently than in the traditional manner, which we have become accustomed too. Increasingly, more and more computation will occur on the Internet rather than just make use of it as a gateway. Programs will be distributed throughout a network and the World Wide Web and work together in a highly developed manner. One method that will increasingly be used to achieve this is mobile agents.

This project endeavored to formulate and develop a method to increase mobile agent execution security, as well as enhance the flexibility and control that a mobile agent platform has over agents and its own platform resources. Due to this, the mobile agent security framework has been successfully designed and a working prototype developed.

The mobile agent security framework is capable of identifying mobile agents and the platforms they have visited using path histories, and delegate system resources that it will be granted access to before it is permitted to execute. The security framework is then used to prevent a mobile agent from accessing any system resources that it is not allocated to use while remaining flexible enough that these chosen system resources can vary depending on a mobile agents' journey.

Mobile agents have a few security vulnerabilities as mentioned in this document, however as these security issues are addressed, mobile agents will become more feasible. As the current research shows, mobile agent security issues are being tackled head on to ensure that they will be just as secure as their client/server counterparts. Mobile agents will provide increased power and performance from the tool we call the information superhighway.

References

1. Henry Chan, Raymond Lee, Tharam Dillon, Elizabeth Chang, E-Commerce Fundamentals and Applications, John Wiley & Sons Ltd, 2001
2. E. Chang, W. Gardner, A. Talevski, E. Gautama, R. Rajugan, T. Kapnoullas, S. Sattar 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th International Conference on Information Systems Analysis and Synthesis (ISAS 2001), Orlando, USA, July 2001
3. E. Chang, W. Gardner, A. Talevski, E. Gautama, R. Rajugan, T. Kapnoullas, S. Sattar. Virtual Collaborative Logistics and B2B e-Commerce, Maximise e-Business ROI Marcus Evans, Duxon Wellington, NZ, 2001
4. Peter Mell, Mark McLarnon, Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems, National Institute of Standards and Technology, Gaithersburg, MD 20899, 1999
5. Wayne Jansen, Peter Mell, Tom Karygiannis, Don Marks, Applying Mobile Agents to Intrusion Detection and Response, National Institute of Standards and Technology, Computer Security Division, NIST Interim Report (IR) – 6416, 1999
6. IBM Alpha Works, XML Parser for Java, September 2002, <http://www.alphaworks.ibm.com/tech/xml4j>
7. Jonna Partanen, Pekka Nikander, Adding SPKI Certificates to JDK 1.2, Third Nordic Workshop on Secure IT Systems, Nordsec, 1998
8. Ernesto Damiani, Sabrina De Capitani, Stefano Paraboschi, Pierangela Samarati, Fine Grained Access Control for SOAP E-Services
9. Pekka Nikander, Lea Viljanen, Storing and Retrieving Internet Certificates, Research Index, 1998
10. Vijay Varadharajan, Security Enhanced Mobile Agents, Distributed System and Network Security Research, School of Computing and IT, University of W.Sydney, Australia, 2000
11. Tuomo Lampinen, Using SPKI Certificates for Authorization in CORBA based Distributed Object-Oriented Systems, Helsinki University of Technology, Department of Computer Science, FI-02015 TKK, Espoo, Finland

An Architecture for Software Agreements between Autonomous Agents

Giles Hogben

Cybersecurity and New Technologies for Combating Fraud Unit
Institute for the Protection and Security of the Citizen
European Commission Joint Research Center
Via Enrico Fermi 1
21020 Ispra, (VA) ITALY
giles.hogben@jrc.it

Abstract. This paper addresses the problem of how to create legally binding software agreements between web services or agents operating within the framework of the semantic web. From an examination of the legal requirements on such a system, a set of formal technical requirements is derived. This then leads to an outline system architecture based on experience in developing preference rule based systems for automated data transfer events.

1 Introduction and Motivation

Currently, no software company would distribute even the most insignificant piece of software without an explicit agreement between the software company and the end user. This is partly to protect intellectual property, but mainly to protect the companies involved against liabilities for the consequences of using their software.

In the scenarios envisaged for Semantic Web technologies, however, there is no clear vision of how legal liabilities can be covered by agreements between autonomous agents or web services, whilst at the same time, there is a clear need for such agreements. The classic Semantic Web scenario envisages an agent performing the entire process of booking a holiday on behalf of a user using a long chain of web services in the process. These services may be dynamically discovered and possibly unknown to the initiating agent, this involves enormous liabilities for agents performing the service on behalf of the user. In classical object brokering scenarios, all objects have been subject to an agreement before any liable actions take place.

Such an agent has to find the best prices, search for trustworthy companies, arrange insurance etc...any of which could be performed by services unknown to the initiating user, somewhere along a long chain of agents or web services. Therefore, beyond the issues involved in getting users to be able to trust such services enough to use them, there are the extremely important issues of legal recourse, liability and consent.

For example, if a pricing service used somewhere in the chain were consistently to give a non-optimal price, some end-users might become aware of this and bring a suit against the company. Companies providing critical information within semantic web goal solution chains therefore need to be able produce formal legal disclaimers, which they can prove have been agreed to by end users and which absolve them of responsibility for such errors.

Such agreements are also necessary for end users. Users need to have clear legal definitions for the terms of use of web-services as well as service level guarantees and within EU law, contact details of the provider responsible for the software [1].

Although some work has been done on the formation of contracts between agents, most notably, by Grosz [2] none has been done on the specific mechanisms for expressing assent or on deriving architectural requirements from explicitly stated legal requirements as is done in this paper.

This paper begins by deriving a set of legal requirements for contract formation between autonomous agents from an analysis of the known requirements and precedents for electronic contract formation in case law and the body of legal documents around click-wrap contracts. From these, a set of general legal principles are set out. From the legal principles and requirements, we derive application requirements and set out a basic architecture which satisfies these requirements.

2 Basis

The basis for the architecture we suggest in this paper is a system we have developed for creating non-repudiable data transfer events [3]. This system uses XML expression of privacy policies and user preference rules. These then provide arguments for a software evaluator which decides on a behavior for the data transfer event.[4]. In this paper I will follow through an analysis of the requirements for an inter-agent software agreement system, and go on to show that a similar system may be used to create reliable agreements between semantic web agents.

3 Formal Requirements for a Semantic Web Enabled Agent to Agent Contract System

In this section, I examine the legal and technical constraints on a contract formed between 2 autonomous agents in order to give a set of formal requirements for the technical construction of a system for forming such contracts.

Throughout the discussion, we will consider the following elements.

L1 A liable entity L1. This is an entity against which sanctions may be taken such as a company or a person as opposed to a software agent.

A1, an autonomous software agent somewhere within a goal fulfillment chain.

C1 a contract, which A1 requires L1 to formally consent to.

S1 a service provider who is the liable entity responsible for A1.

The definition of such a system is determined by a set of legal requirements, which must be fulfilled and take precedence over technical considerations. Therefore in order to obtain the specific requirements on an agent-agent contract formation system, we first define the legal requirements of the system. These come from an analysis of “traditional” online contract law:

4 Legal Requirements

The following is a formal list of requirements, derived by examining the body of documents on traditional electronic contract formation. Exact quotes are given where possible but sources can be found in both European and US law.

Legal Requirement 1.

Assent to a contract can only be provided if the party giving assent has been given reasonable opportunity to review and comprehend the terms of the contract.

"The information... the commercial purpose of which must be made clear, shall be provided in a clear and comprehensible manner in any way appropriate to the means of distance communication used, with due regard, in particular, to the principles of good faith in commercial transactions." [1]

"[Assent] can occur only if the party had an opportunity to review the terms prior to assenting"[1]

Legal Requirement 2.

It is the duty of the end-user to review the contract, whereas it is the duty of the service provider to provide it in a comprehensible and accessible manner. [5]

In legal terms, the concept of comprehensibility referred to above, may traditionally refer to a *mens rea* - that is a provable mental state. In the case of a software agent, however, this meaning of the term cannot be used. Also it is clear from case law that it is more the *comprehensibility* and accessibility of the contract that is the issue than the actual act of comprehension

Legal Requirement 3.

The contract must be available for review by a reasonable and unambiguous mechanism.

This point is clear from the Specht v. Netscape case [6] where the Court held that an act of downloading software did not indicate assent to be bound by terms of a license agreement, because a link to the terms appeared on, but below that portion of the web page that appeared on the user's screen when the download took place. This is also clear from [5].

Legal Requirement 4.

Both the software agent and the service must carry the mandated responsibility of a human being.

The purpose of any contract is to provide a bi-lateral agreement, which can be enforced within a legal system. This cannot occur unless the agreement is between 2 parties, which have legal liability. As a software agent cannot be punished and retribution cannot be carried out against it, unless the software agent can be made to carry the mandated responsibility of a human being, such agreements are not possible.¹

Legal Requirement 5.

For a contract to be comprehensible, L1 must understand unambiguously the language it is written in.

Again – the traditional application of the concept of understanding as a *mens rea* cannot be applied.

Legal Requirement 6.

Assent must be actively and unambiguously expressed.

Assent may be expressed "by conduct" where actions may be given assertive meaning by an agreed protocol.

(Vide *Hotmail Corporation v. Money Pie Inc* [7])

The essential legal requirements can be summarized

- Comprehensibility and opportunity for review.
- Unambiguous assent "by conduct".
- A clear mechanism of mandate.

All these requirements will be relevant in the discussion below. However, the most important points are that A1 must possess some form of evidence of responsibility mandated by L1 and that it is required for the contract to be "comprehensible" by A1.

The question of comprehensibility is perhaps the most problematic because, as noted, it may refer to a mental state, which cannot, with its most common meaning, be attributed to a software agent.

To approach this problem, however, we consider that the concern of the law is essentially that the commitments contained in the contract do not go against the wishes of the person assenting to it. In demanding that the contract is comprehensible, the law is, in practical terms, demanding that the user has had the opportunity to evaluate the contract against his/her value set.

5 Derived Principles

From the above legal requirements, we can distil the following core principles which must apply in the case of contract formation between autonomous agents:

¹ It is even possible within legal frameworks which do not specify retribution as the purpose of punishment, to envisage systems for "punishing" agents. However, as this possibility is for the moment highly fanciful and is unlikely to be included within any legal framework for many years, we will ignore it.

Principle 1.

If a Liable Entity (L1) can express his/her value set wrt online contracts, in a machine readable format, then an autonomous agent, A1 can for legal purposes be said to comprehend a contract if it is able to evaluate any contract written within the same semantic standard, in such a way that it will never agree to a contract which L1 would not agree to if he/she were to read and comprehend the contract.

It is impossible to imagine how an agent might function without a model of the preferences in regard to software licenses the user is consenting to. Without such a model, the user would have to rely on a behavior model designed by a third party. This would be illegal unless the user had in some way consented to the preference model. A parallel can be seen in models of privacy preferences, where the actual mechanics of the preferences are too complicated for the ordinary user to specify, so standard sets are provided. They are still the preferences of the user, because at some point, the user makes a deliberate choice between the different sets.

"Comprehend" or "understand" for autonomous agents may only be understood in the behaviorist sense of the Turing Test [8]: if A1 behaves in exactly the same way (barring the application of maximum precaution) as if it understood the contract and preference rules in the same way as L1. I.e. - A1 will never agree to a contract which L1 would not agree to if he/she were to read and comprehend it. A1 then models the course of maximum precaution for L1's behavior. Note that A1 may not even be known to L1, but may be mandated a task by a primary agent which is used by L1.

Principle 2.

If A1 is mandated to assent to a contract by L1, then A1 would not assent to anything that L1 would not assent to.

Following from the definition of comprehension as the ability to evaluate a contract against a user's XML preferences, we need to introduce this strong precautionary requirement in order to prevent breaches of trust.

Principle 3.

Assent can be expressed by the agent performing an action, which, according to an approved standard has the unambiguous meaning that it assents to the contract and providing evidence that it has the mandate of the user to do so.

This derives from the concept of the "click-wrap" or "point and click" contract which uses "assent by conduct". A certain user action such as pressing a given button, is clearly designated as having the meaning of asserting that the user has understood and agrees to something. The designation is in a form that the user understands by default. The action must be something that the user would not do accidentally. The crucial point is that the user in the case of a "click-wrap" contract can have no other reason to click the button other than to assert he has understood the terms of the contract.

6 Formal Requirements for a Semantic Web Agent-Agent Agreement System

Using the above legal requirements and derived principles, we can now state the following requirements of a system for provision of contracts between agents.

Application Requirement 1.

A1 must have available a set of machine understandable preference rules, R.

Application Requirement 2.

R must be expressed in a standard, machine understandable, validatable syntax (e.g. XML [9]+validation schema), which is universally agreed on by the legal and technical community, and by L1, so that L1 cannot argue ambiguity of terminology and it is usable by A1. (From Principle 1.)

Application Requirement 3.

R must be expressed in such a way that A1 would not assent to anything that L1 would not assent to. Given that L1 cannot understand XML, this then has to refer to the semantics of any agreement rather than its particular form (*ad substantiam actus*). (From Principle 1.)

Application Requirement 4.

There must be some proof (e.g. a digital signature or weaker) that A1 is mandated by L1 to use R (i.e. that L1 has given consent to A1 to act according to R). If this is not required, agents could sign contracts on behalf of human beings who have not mandated them with any responsibility. The level of this proof will depend on the level of the liabilities being entered into (for example digitally signed mandate required if a financial transaction > \$500). (From Legal Requirement 4.)

Application Requirement 5.

L1 must be responsible for making sure that A1 is able to "understand" R and C1, in the sense of understand described in . (From Legal Requirement 2. and Principle 3 (precaution))

Application Requirement 6.

C1 must be available to A1 by means of a standardized and unambiguous discovery mechanism. (From Legal Requirement 3.)

Application Requirement 7.

C1 must be expressed in a standard, machine understandable, validatable syntax, X2 (e.g. XML + validation schema), which is universally agreed by the legal and technical community, so that L1 cannot argue ambiguity of terminology and it is usable by A1. (From Legal Requirement 5.) This syntax must be consistent with R in that semantic elements in C1 correspond in an agreed way with those in R, so that A1 may accurately evaluate C1 against R.

7 Implementation and Example

The process envisaged here may be compared to the process used in W3C's P3P protocol [10], where companies publish XML privacy policies expressed in standardized vocabulary. End users express privacy preference rules in a related vocabulary, and an automated decision is made on whether to make requests involving data transmission or not. In this case, companies make available contracts expressed in standardized RDF/xml [11] vocabulary and end users express their preferences with regard to contracts in a related vocabulary. Automated agents are then able to make decisions on whether to agree to the contract or not according to a standardized, legally valid protocol whose behavior is reliable and can be determined precisely.

For the purposes of illustration, consider that a contract system ontology, defined using DAML 2001, which defines only

1. The classes: "web service", "return value" and "end user":

```
<daml:Class rdf:ID="web service">
<rdfs:label>web service
</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="return value">
<rdfs:label>return value</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="end user">
<rdfs:label>end user</rdfs:label>
</daml:Class>
```

2. The properties: "may resell" and "is a return value of".

```
<daml:ObjectProperty rdf:ID="may resell">
<rdfs:label>may resell</rdfs:label>
<rdfs:domain>
<daml:Class rdf:about="end user">
</daml:Class>
<daml:Class rdf:about="web service">
</daml:Class>
</rdfs:domain>
<rdfs:range>
<daml:Class rdf:about="return value">
</daml:Class>
</rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="return value of">
<rdfs:label>return value of</rdfs:label>
<rdfs:domain>
<daml:Class rdf:about="return value">
</daml:Class>
```

```

</rdfs:domain>
<rdfs:range>
<daml:Class rdf:about="web service">
</daml:Class>
</rdfs:range>
</daml:ObjectProperty>

```

For the purposes of the example, we state that rule system can use only the classes and properties in the rule head (the part of the rule which has a condition - "if you find xyz then . . .") The rule would consist of an RDF query and a behavior. If the RDF query produces one or more results then the behavior is activated, if not, then the next rule in the set is executed. Within this system then you could write a useful set of 2 rules:

1. Match rules in the policy of type (Notation: N3[12]):

Log:forall :x, :y

(CSO:end user :x CSO:may resell (CSO:return value :y where :y CSO:return value of :web service :z))

Behavior:Assent

In plain English, this means

If the agreement states that all end-users may resell the return values of the service then assent to the agreement.

2. A catch all rule,

Match: ()wildcard*

Behavior:Do Not Assent

In plain English: *If all other rules do not fire, do not assent.*

In this somewhat artificial environment, then, the end-user could be sure that nothing could be agreed to which did not conform to this preference. Obviously this example is too restricted to be useful but it shows how such a rule logic may be applied in such a way as it can be relied upon not to compromise the user's preferences, and an extension of this type of rule implementation would produce a useful as well as reliable preference model.

8 Implementation Architecture

- An ontology for software contracts. We suggest using a process such as that proposed by Hameed [13]; The ontology should incorporate a flexible model to allow for extensions. The ontology should use the OWL [14] syntax. This component of the architecture is perhaps the most difficult to implement as it requires a consensus between technical implementers and lawyers.
- Preference rule matching language. The language would follow an ECA (event, condition, action) format.

- A Graphical User Interface, which allows the user to specify patterns or characteristics to look for within contracts. It is inevitable that the patterns the user is to look for will be somewhat stereotyped. However, with care, as we have found in the case of APPEL [15], the privacy preference rule matching language, functional interfaces can be created. In particular, sets of default standardized preference sets may be created by experts, characterizing typical preference profiles and then distributed within software applications for end-users.
- A matching engine, which looks for these patterns, specified by the user
- A behavior manager, which based on the characteristics matched in an RDF contract, decided what action to take. This will generally be a simple binary choice, but this component could conceivably be extended to negotiation, asking for a digital identification from the service provider or other behaviors.
- Contract creation interface. This would be a GUI linked to the contract ontology, which would allow service providers to create XML contracts. Again, a default set of standard contracts might be able to cover many cases.

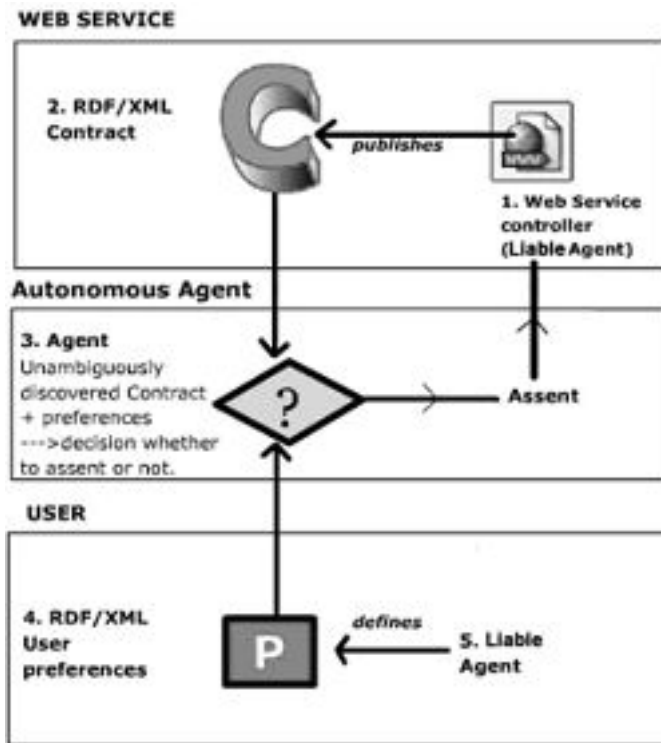


Fig. 1. Architecture scheme for semantic web enabled contract formation.

The following supporting components are also required

- Contract discovery module. To discover whether or not a service requires the signing of a contract, and if so, where the XML for that contract is located and precisely how consent should be expressed.
- A module for either signing contracts or otherwise unambiguously expressing consent. Such a signature or proof should contain proof of the mandate of L1. Note that if the service is set up such that it cannot be accessed without a certain action being performed by the agent (assent by conduct), then there is no need for the service to store consent.
- Contract verification module. A module for verifying that contracts are approved and published and approved either by the issuing service or by a trusted third party or both.

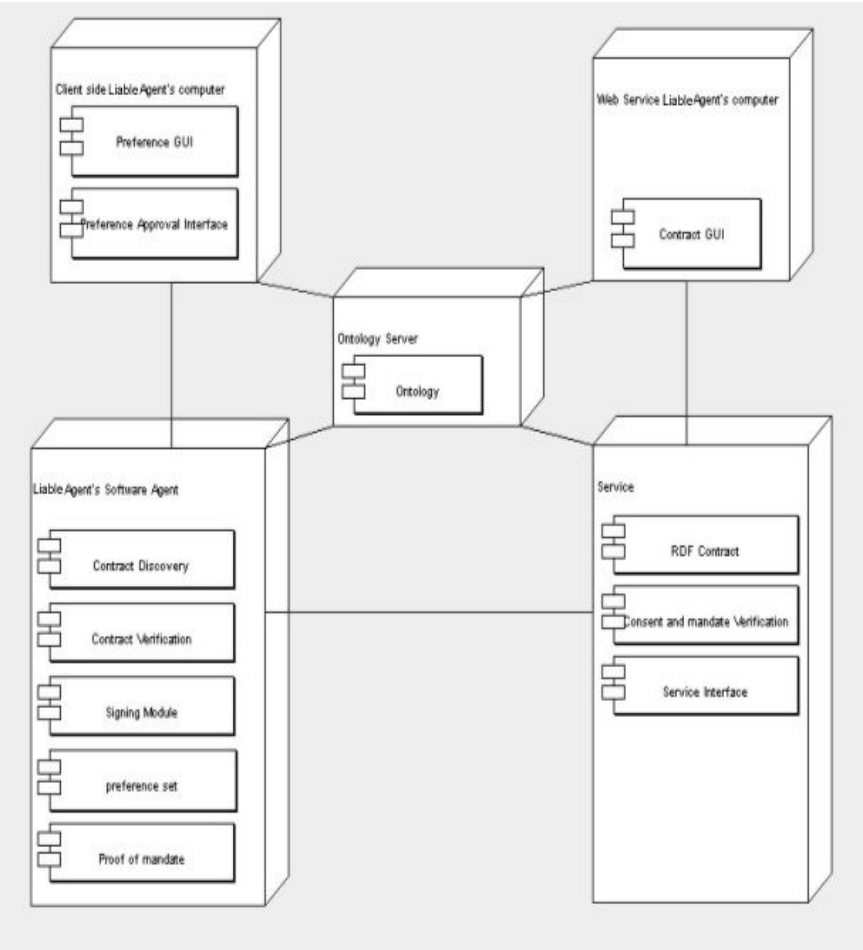


Fig. 2. UML Deployment Diagram Showing Principal Components and Relationships

9 Conclusion

By examining legal requirements for click-wrap contracts, we have derived a set of requirements for formation of contracts between autonomous agents which implements a precautionary principle to avoid excess liability. We suggest an architecture for implementing these requirements based on an ontology and rule language which can provide a basis for the expression of contract agreement rules which are able to protect the liable party while still allowing sufficient flexibility to be practical.

References

- [1] DIRECTIVE 97/7/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 20 May 1997 on the protection of consumers in respect of distance contracts).
- [2] Grosz, B. Poon, T. MIT Sloan School of Management, Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions
- [3] Vide Hogben, G., Wilikens, M., Jackson, T: A Fully Compliant Research Implementation of the P3P standard, Proceedings of 7th European Symposium on Research in Computer Security, , Springer Verlag
- [4] Implementation downloadable from <http://p3p.jrc.it>
- [5] US Law, UNIFORM COMPUTER INFORMATION Act
- [6] See "Specht et al. v. Netscape Communications Corp. and America Online, Inc." (2001 WL 755396 S.D.N.Y., July 5, 2001).
- [7] Hotmail Corporation v. Van Money Pie Inc., et al., C98-20064, 1998 WL 388389 (N.D. Ca., April 20, 1998). For a description of this case, see <http://www.phillipsnizer.com/int-art62.htm>
- [8] Turing, A.: 'Computing Machinery and Intelligence', *Mind* 59(236) (1950) pp. 433–460
- [8] Extensible Markup language W3C standard. <http://www.w3.org/XML/>
- [9] The Platform for Privacy Preferences 1.0 (P3P1.0) Specification W3C Recommendation 16 April 2002 <http://www.w3.org/TR/P3P>
- [10] See <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>
- [11] N3 - Notation3 language for expressing RDF triples in readable format <http://www.w3.org/DesignIssues/Notation3>
- [12] Hameed, A., Sleeman, D., Preece, A.: University of Aberdeen [2002]. Reconciling Experts' Ontologies for the Semantic Web. Proceedings of the First International Semantic Web Conference (ISWC-2002), Sardinia, Italy, June 9–12, 2002, Springer Verlag
- [13] See <http://www.w3.org/2001/sw/WebOnt/>
- [14] See <http://www.w3.org/TR/P3P-preferences/>

WRSM 2003 PC Co-chairs' Message

This Workshop aims to integrate four important computer system-engineering technologies: reliability, security, middleware and agent-based systems. WRSM 2003 was intended to be a selective Workshop for fostering collaborations amongst leading international researchers in the fields of reliable and/or secure middleware, and agent-based systems. This year's technical program fulfils that expectation and, in fact, breaks new ground again as the themes of WRSM 2003, and its focus on cutting-edge research and technologies, are all the more timely given the recent events in the world.

The program for WRSM 2003 consists of fifteen research papers of very high quality, covering a breadth of diverse topics, such as fault-tolerance, quality-of-service, agent-based middleware and enterprise computing issues. Recognizing that the graduate students of today are our future scientific leaders of tomorrow, WRSM 2003 incorporates a special Student Research Workshop session, where graduate students are encouraged to present their latest findings, to receive feedback from senior researchers, and to gain valuable exposure, even at the early stages of their careers.

WRSM 2003 is continuing the long-standing tradition of workshops, by serving as a forum for healthy technical debate, and for the exchange of newly recognised research issues, promising advanced formulations, and research progress reports.

We are grateful to the members of the WRSM 2003 Technical Program Committee for reviewing the papers, for helping us to assemble such an outstanding program, and for their efforts in making WRSM 2003 a memorable technical event. We extend our appreciation to the organizers of the On The Move Federated Conferences 2003 for their valuable advice and support. Finally, we would like to thank our authors for submitting publications of such high quality, and for sharing the results of their research work freely with the rest of this community.

August 2003

Priya Narasimhan, Carnegie Mellon University, USA
Brian Blake, Georgetown University, USA
Giacomo Cabri, University of Modena, Italy
(Workshop on Reliable and Secure Middleware 2003
Program Committee Co-chairs)

Fairly Redistributing Failed Server Load in a Distributed System^{*}

Venkatesh Sangam, Christopher B. Mayer ^{**}, and K. Selçuk Candan

Computer Science and Engineering Department
Arizona State University
{venkatesh.sangam,chris.mayer,candan}@asu.edu

Abstract. We recently proposed a novel method for large-object replication and load balancing. Our method is particularly well-suited to data grids, data warehousing providers, and hosting of dynamic web sites. The method attempts to distribute object request load fairly to servers according to server capacity so that the likelihood of them overloading, and hence failing, is reduced. Unfortunately, server failures cannot be eliminated entirely. When a server fails, the load carried by that server must be absorbed by the rest of the system. Unless this load is distributed fairly across the remaining servers, they may also overload, creating a cascade of failures and reduced quality of service. In this paper, we propose an efficient method for fairly redistributing the load of a failed server or set of failed servers within our replication system. We also report on experimental results that verify the validity of our approach.

1 Introduction

We are investigating the replication of large data objects in distributed environments. Such environments include data grids, data warehousing services, and dynamic web site hosting. Individual data grid members (typically research institutions) must maintain data generated by their own endeavors and be ready to share that data with other data grid members [7]. Data warehousing services archive data from numerous sources. Typically this data is stored as views that are made available to the warehouse's clients. Dynamic web site owners outsource their sites to hosting services in order to obtain an instant Internet presence and economies of scale. The hosting service maintains copies of the databases and application logic constituting a dynamic web site on its servers.

In order to ensure availability and meet user demand, replication of the data (and any needed application logic) in each of these example environments may be required. These environments share several properties which must be considered in any replication solution.

^{*} This research funded by NSF grant 998404-0010819000.

^{**} The views expressed in this paper are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

- **The basic unit of data access is on the order of megabytes.** For example, the basic unit of access for a data grid is an experiment or a series of data collection efforts, a view for a warehouse, and the database for a dynamic web site. Data size precludes rapid, on-demand replication. Thus content needs to be pre-positioned in anticipation of demand.
- **Data objects number in the hundreds or low thousands with new objects being introduced infrequently.** Since we are dealing with a low number of infrequently changing objects, it becomes possible to locate servers with desired objects using a directory service.
- **Replication may be implemented as a distributed system.** The replication system may use dozens or hundreds of servers located in multiple buildings or scattered across the globe. Servers may vary in terms of performance.
- **Servicing user requests is the main contributor to server load.** While updating data on a server is also a source of load, updates are infrequent and can be effectively ignored. For our purposes, a server's load is only its read load – the load of servicing user requests.
- **Response times are high.** In the case of data grids and warehouses, transmitting megabytes of data dominates the response time. For dynamic web sites, processing a user's query through the database is the main response time component [9].
- **Server load is a key concern.** Transmitting large amounts of data or performing database queries uses a significant amount of server capacity. Servers become sluggish and can appear to fail when they become overloaded. The main way to ensure system responsiveness is to control server load. Controlling responsiveness by placing objects close to users (see [1,2,5,6,8] for examples) is not as important.
- **There is an authority that can dictate the rules with which various data objects are replicated.** For instance, a web data hosting service can decide how its clients' sites will be replicated on its many servers and how the requests to those replicas will be directed.
- **The authority has no control over which objects user will access nor which server users will contact when initiating a request.** Also, we cannot assume that users have any knowledge of the replication system's inner workings. Therefore, the authority must direct requests to the most suitable replica server in a manner transparent to the users. As with many distributed systems, each user has a regular set of entry points (servers) from which they access the system.

In a previous work [3], we proposed an architecture and design method for replication systems for large object environments such as those mentioned above. While it addresses the issues listed above, our proposal also focuses on distributing system read load to servers in proportion to each server's capacity. That is, a server should expect to shoulder the same fraction of system load as it contributes to system capacity. Such a server is said to be **fairly loaded**. The advantages of a load-fair system are twofold. First, systems can use servers of

varying capacities. Secondly, server load tracks system load so that it is unlikely that a server will overload (temporarily fail) unless the system is approaching its capacity limit. In [3] we showed how to construct highly load-fair systems when all servers are operating. We will review our proposed architecture later.

In this paper we extend the work of [3] by showing how to maintain load fairness in the face of server failures. In a load-fair system server failures can be extremely detrimental. A failed server means lost system capacity and may make certain system services or data unavailable, which is bad enough. Worse though, the load normally carried by a failed server must be distributed to other servers in order to maintain quality of service. To reduce the likelihood of more failures, the failed server's load should be redistributed to other servers so that load fairness is maintained.

The paper is organized as follows. Section 2 contains a review of our replication system architecture. In Section 3 we explain how failures affect our replication system. Section 4 describes how to maintain fairness when servers fail. The effectiveness of our failure handling strategy is demonstrated in Section 5. Section 6 summarizes our efforts.

2 Replication System Background

Our replication system organizes a system's servers, $S = \{s_1, s_2, \dots, s_n\}$, into sets called write-sets, W , and read-sets, R . Replicating a data object is done by selecting a write-set and copying the object to all the servers in it. To access an object or perform a query, a read-set is selected and an appropriate server from that read-set is selected to perform the requested action. To ensure that user requests can always find a server, every write-set must intersect every read-set. The system implementer (data grid member, warehouse service, or hosting service) decides to which write-sets an object should be placed and relocates objects in order to balance load across the write-sets.

Since each server in the system can have a different capacity, forming write-sets and read-sets that promote load fairness can be difficult. To make it easier, we transform each server into a number of **virtual servers** as follows. For each $s_i \in S$, turn s_i into n_i virtual servers by first dividing s_i 's data transmission capacity, C_i , by a capacity, C_{base} , that is less than or equal to the capacity of the lowest capacity server in S and then compute the floor of the result, i.e., $n_i = \lfloor \frac{C_i}{C_{base}} \rfloor$. The set of s_i 's virtual servers is $V(s_i) = \{s_{i,1}, s_{i,2}, \dots, s_{i,n_i}\}$. Notice that each virtual server has about the same amount of capacity. Instead of forming write-sets and read-sets from physical servers, we form them from virtual servers. Note that virtual servers are instances of the same physical server. Each virtual server of a physical server has access to the same content as its physical server. The content on a physical server is determined by its membership in write-sets via its virtual servers.

In [3] we used a simple grid structure for establishing write-sets and read-sets, where the l rows of a grid become the write-sets and the k columns are

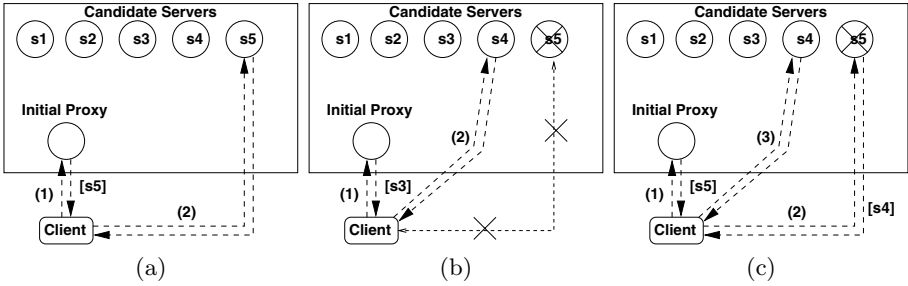


Fig. 1. Examples of replication system operation. In (a) there are no failures. In (b) server *s*₅ has hard-failed and the proxy must act as the failure-handling entity. In (c) *s*₅ is soft-failed and acts as its own failure-handling entity by redirecting requests sent to it by a proxy to other servers.

the read-sets.¹ We showed how to map virtual servers, one per grid cell, to form write-sets and read-sets that promote fairness. Using this structure, each virtual server appears in only one write-set and one read-set. However, because a server can have many virtual servers, a server can be in multiple write-sets and read-sets. We also introduced the following generic read protocol.

1. The user selects a server (a **proxy**) and sends an **initial-request** indicating the content desired to the proxy.²
2. The proxy selects a read-set according to a probabilistic **proxy strategy**.
3. The proxy identifies the server(s) in the selected read-set capable of handling the request by consulting directory service or a lookup table. If more than one server is capable, one of them is picked equiprobably.
4. The proxy redirects the request to the selected server.
5. The selected server processes the request and returns a response to the user.

This read protocol is illustrated in Fig. 1(a). In step 1 a user (client) contacts a proxy with a request. The proxy then selects a read-set (candidate servers) and determines which server should handle the request, in this case *s*₅. The proxy redirects the user to *s*₅ and *s*₅ fulfills the user's request (step 2).

Note that proxies actually select virtual servers during the read protocol. A virtual server's "load" is determined by its selection by proxies. Since a server's load is the sum of the "load" of its virtual servers, a server's load is a factor of the proxy strategies and the structure of the write-sets and read-sets.

Using this read protocol and write-sets and read-sets formed from virtual servers, we showed in [3] how to formulate proxy strategies that, given a pattern of initial-requests to the proxies, distributes load fairly to read-sets and

¹ Although we used a grid, almost any logical structure can be used. The method for promoting fairness depends on the structure. For this paper, what is important are the write-sets and read-sets and that they intersect.

² While we consider proxies to be a server which contains replicated content, the proxy may exist solely to direct requests.

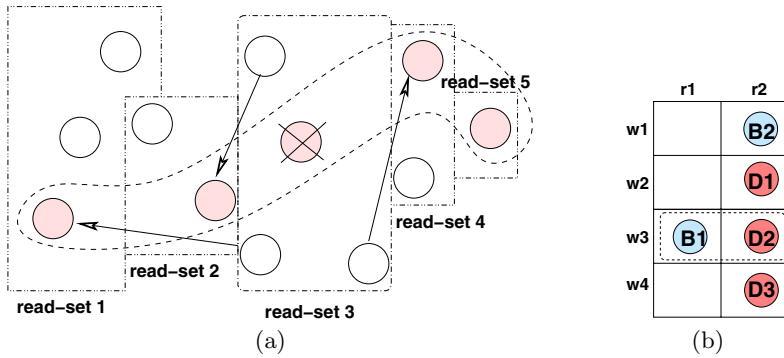


Fig. 2. A failure example using write-sets and read-sets. In (a) a failed server (marked with an X) causes servers in read-set 3 to use servers outside the read-set to access content in the write-set (the servers contained inside the dashed line). In (b) virtual servers allow redirection within a read-set.

ultimately to the system's servers. In [4], we examined the sensitivity of our approach to variations in user requests arriving at the proxies.

3 Fairness and Failures

A fair replication system distributes system read load to its member servers such that the fraction of load each server experiences is equal to its contribution to the system's capacity. Equivalently, the loads experienced by any two servers should be roughly equal to the ratio of their capacities. Modeling a replication system as a set of servers, $S = \{s_1, \dots, s_i, \dots, s_n\}$ where each server has a known amount of serving capacity, C_i , the **fairness condition** to be enforced can be written as:

$$\forall s_i, s_j \in S, \frac{L(s_i)}{L(s_j)} \approx \frac{C_i}{C_j}. \quad (1)$$

Replication systems built and operating as described in [3] have a high degree of fairness when all servers are operating. However when a server fails, the system becomes unfair causing more servers to fail due to overloading.

When a server fails, another server needs be selected to serve the user's request. Since servers in a write-set have the same content, this **replacement server** should come from the same write-set as the failed server. Selecting a replacement server has to be done quickly so that response times remain low. Of course, we do not always know that a server has failed before we attempt to use it. Requests sent to a failed server will unavoidably have poor response times. However, once detected, we can avoid the failed server and keep response times low by directing requests only to live servers.

One way to work around a failed server is to avoid selecting read-sets with failed servers. However, the proxy strategies for selecting read-sets are meant to

ensure system-wide fairness. If we were to suddenly start avoiding read-sets with failed servers, system-wide fairness could be severely affected. A better solution is to continue selecting read-sets according the proxy strategies, but to select other servers within the read-set *or* modify the read-set's membership to include servers with the same content as the failed server. Thus we can avoid paying time and fairness penalties associated with choosing another read-set.

Virtual servers allow write-sets and read-sets to intersect at multiple servers. Intersecting at multiple servers decreases the need to modify a read-set's membership. For example, in Fig. 2(a) there are no virtual servers. When a server fails (the crossed-out server), requests arriving at servers in the failed server's read-set must be redirected to servers outside the read-set. However, if virtual servers are used, redirection can take place within the read-set and no modification is needed. Figure 2(b) shows an example of this. Here, write-set $w3$, which contains virtual servers of B and D , intersects read-set $r2$ with four virtual servers. If D fails, B can still service requests directed to $r2$ for objects stored on $w3$.

Suppose that a system using virtual servers is load-fair when no servers have failed. While write-sets and read-sets can intersect with multiple servers, a failed server may still cause an imbalance in load fairness. Also, it may not be possible to ensure such intersections in all cases. Thus, we must consider modifying a read-set's membership to have a better chance of keeping the system fair.

4 Preserving Fairness When Servers Fail

Above, we made the case for introducing new servers into a read-set to replace a failed server and to preserve fairness. In this section we introduce a method for selecting these replacement servers in a way that promotes load fairness for the remaining live servers. We tackle this in a number of steps. First, we list driving concerns that the method should meet. Next, we introduce the concept of replacement sets and then state the constraints that need to be met in order to ensure fair load distributions. We then present a technique for maintaining fairness when servers fail, first for the simple case of single-server failures and then for the more complicated case where several servers have failed simultaneously. Finally, we discuss the information needs and responsiveness of our failure handling technique.

4.1 Driving Concerns

The way in which a read-set is modified can have a big impact on the performance (in terms of cost, response time, failure probability etc.) of the overall system. In addition to maintaining fairness, there are the following concerns.

- Creating the re-routing plan and the actual re-routing of requests should both be fast.
- Since the system might be distributed, the changes to the routing policy should be computed in a distributed fashion, too. The entity responsible for

redirecting requests around a failed server, the **failure-handling entity**, should use mostly local information.

- The failure-handling entity can be located at many places within the system.

To further motivate our approach for preserving fairness and to see how these concerns relate, consider Fig. 1 which shows two possible failure-handling entities and how they deal with the failure of server $s5$. In both parts of the figure, $s5$ has been selected by the proxy for handling a user's request. In Fig. 1(b) $s5$ has “hard-failed” and is totally unresponsive. Someone else, such as the proxy, must handle request redirection. Any requests sent to $s5$ before the proxy learns of $s5$'s failure will have enlarged response times. In Fig. 1(c) $s5$ has “soft-failed”; it has communication capabilities but cannot fulfill the user's request (perhaps the disk with the needed data is down). Knowing its problem, $s5$ acts as its own failure-handling entity and redirects the user's request to server $s4$. A server, like $s5$, that is its own failure-handling entity saves other system components the burden of redirecting requests. Also, since the soft-failed server is immediately aware of its problems, the lag time in detecting and adapting to failures can be reduced. Thus, we see the advantages of having multiple locations for the failure-handling entity. We also see an overall need for fast re-routing of requests.

4.2 Replacement Sets

For each virtual server in the system, we can define a **replacement set**. A replacement set consists of all the servers that appear in the virtual server's write-set. Since (i) all servers in a given write-set have the same objects and (ii) a server is in a write-set if it has a virtual server the write-set, only the servers in the replacement set can be replacement servers. Replacement sets conform nicely with our read protocol. A proxy can select a read-set and a server (via a virtual server) within that read-set as the read protocol states. If the selected (virtual) server has failed, the request can be redirected to a server in the virtual server's replacement set. The example below illustrates the replacement set concept.

Example 1. We have a small replication system using six servers A , B , C , D , and E . The servers have the following capacities: $C_A = 4$, $C_B = 4$, $C_C = 2$, $C_D = 3$, and $C_E = 3$. The number of virtual servers for a server is equal to the server's capacity. Using a grid to define write-sets and read-sets, a possible mapping of virtual servers is shown below.

A1	B1	C1	D1
B2	A2	A3	E1
E2	B3	D2	C2
D4	A4	B4	E3

Let us assume that server E fails. E appears in the second, third, and fourth rows (write-sets) and first and fourth columns (read-sets). Using $R(x)$ to denote the replacement set for virtual server x , the replacement sets for E 's virtual servers are: $R(E1) = \{A, B\}$, $R(E2) = \{B, C, D\}$, and $R(E3) = \{A, B, D\}$. $\diamond$

Unless replacement sets are used carefully, redistribution of load may not be fair. For instance, in the above example, server B would get a higher portion of E 's load than server A if the load is uniformly distributed within the replacement sets since B is in all three of E 's replacement sets versus two for A . However, both servers have the same capacity (4 each). To be fair, they should share an equal portion of E 's load. In the next section, we formulate constraints that ensure that the load of the failed servers is redistributed fairly.

4.3 Fairness and Replacement Sets

Say server s_f fails. To fairly distribute its load, the load s_f would normally have carried must be directed to the servers in its replacement sets in proportion to their capacities. Below we list conditions that, when satisfied, ensure a fair distribution. Before describing these conditions, we need to define some terms and introduce some notation. Let

- $S = \{s_1, s_2, \dots, s_n\}$ be a set of servers, where each server s_i is split into a set of n_i virtual servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \dots, s_{i,n_i}\}$;
- C_i denote the serving capacity of server s_i ;
- $w(s_{i,j})$ be the write-set containing the j^{th} virtual server of server s_i ;
- $R(s_{i,j})$ be the replacement set for the j^{th} virtual server of server s_i such that

$$R(s_{i,j}) = \{\{s_k | s_k \in w(s_{i,j})\} - s_i\}$$

- (i.e., $R(s_{i,j})$ contains servers in the same write set as $s_{i,j}$ except for s_i itself);
- $R(s_i) = R_{i,1} \cup R_{i,2} \cup \dots \cup R_{i,n_i}$ be the set of replacement servers of server s_i ;
- $L(s_{i,j})$ be the load on the j^{th} virtual server of server s_i ;
- $X(s_i)$ be the **extra load** on server s_i due to directing requests from failed server s_f ($s_i \neq s_f$) to s_i .

The constraints for fairly distributing the load of failed server s_f are as follows.

- **Constraint 1:** Servers in s_f 's replacement set get a fair share of s_f 's load

$$\forall s_i, s_j \in R(s_f), \quad \frac{X(s_i)}{C_i} = \frac{X(s_j)}{C_j}. \quad (2)$$

- **Constraint 2:** The amount of extra load replacement server s_i receives depends on the load of each the failed server's virtual servers and the **redirection probability**, $p(s_{f,j}, s_i)$, of sending requests from each of those virtual servers, $s_{f,j}$, to s_i

$$\forall s_i \in R(s_f), \quad X(s_i) = \sum_{s_{f,j} \in V(s_f)} p(s_{f,j}, s_i) \cdot L(s_{f,j}). \quad (3)$$

- **Constraint 3:** All of a failed virtual server's load must be redirected to servers in its replacement set

$$\forall s_{f,j} \in V(s_f), \quad \sum_{s_i \in R(s_{f,j})} p(s_{f,j}, s_i) = 1.0. \quad (4)$$

Next, we show how to use these constraints (Equations 2 – 4) to calculate redirection probabilities for single-server and multiple-server failure scenarios.

Let s_f be the failed server

1. Get the replacement sets for s_f .
2. Calculate redirection probabilities for each server in the replacement set using Equations 2 – 4.
3. When one of s_f 's virtual servers, $s_{f,j}$, is selected for handling a request, substitute a server in the replacement set of $s_{f,j}$ based on the redirection probabilities.

Fig. 3. Pseudocode for calculating redirection probabilities.

4.4 Calculating Redirection Probabilities

The algorithm in Fig. 3 outlines how to calculate redirection probabilities. Example 2 demonstrates how the algorithm might be employed for a simple single-failure scenario.

Example 2. Using the system and replacement set from Example 1 let

- $p(E1, A)$ and $p(E1, B)$ denote the redirection probabilities for servers A and B for $E1$'s replacement set $R(E1)$,
- $p(E2, B)$, $p(E2, C)$, and $p(E2, D)$ denote the redirection probabilities of servers B , C , and D for replacement set $R(E2)$, and
- $p(E3, A)$, $p(E3, B)$ and $p(E3, D)$ denote the redirection probabilities for servers A , B , and D for replacement set $R(E3)$.

Via (4) we have:

$$\begin{aligned} p(E1, A) + p(E1, B) &= 1.0 \\ p(E2, B) + p(E2, C) + p(E2, D) &= 1.0 \\ p(E3, A) + p(E3, B) + p(E3, D) &= 1.0. \end{aligned}$$

Using (3) we can express the extra load on each server due to the failure of E as

$$\begin{aligned} X(A) &= p(E1, A) \cdot L(E1) + p(E3, A) \cdot L(E3) \\ X(B) &= p(E1, B) \cdot L(E1) + p(E2, B) \cdot L(E2) + p(E3, B) \cdot L(E3) \\ X(C) &= p(E2, C) \cdot L(E2) \\ X(D) &= p(E2, D) \cdot L(E2) + p(E3, D) \cdot L(E3) \end{aligned}$$

Using the read protocol and assuming objects in each write-set are equally popular and that read-sets are selected uniformly, we use the methods of [3] to determine that the fraction of system load on each of E 's virtual servers is $L(E1) = 0.08125$, $L(E2) = 0.05347$, and $L(E3) = 0.08125$. To balance the extra load fairly, (2) has to be satisfied, so we write

$$\frac{X(A)}{C_A} = \frac{X(B)}{C_B} = \frac{X(C)}{C_C} = \frac{X(D)}{C_D}.$$

Solving these equations gives numbers for the redirection probabilities which ensure a fair distribution of load: $p(E1, A) = 0.1821$, $p(E1, B) = 0.8179$, $p(E2, B) = 0.0$, $p(E2, D) = 0.6214$, $p(E2, C) = 0.3786$, $p(E3, A) = 0.6358$, $p(E3, B) = 0.0$, and $p(E3, D) = 0.3642$. $\diamond$

When multiple servers fail, the calculation of redirection probabilities is more complicated for two reasons. First, replacement sets may contain failed servers. Since a failed server cannot act as a replacement server, failed servers have to be removed from all replacement sets of which they are members. Secondly, newly failed servers and any failed server that has a newly failed server in one or more of its replacement sets need to have their redirection probabilities (re)calculated. We demonstrate how to handle multiple server failures in the following example.

Example 3. We continue Example 2 where E was the only failed server. Suppose server A also fails. A 's failure means that E 's replacement sets and redirection probabilities need to be recomputed. Since the extra load that can be carried by server A in the replacement set of E is now zero (i.e., $X(A) = 0.0$) we need to set the redirection probabilities for server A to zero by making $p(E1, A) = p(E3, A) = 0$. The load and redirection constraints become, respectively,

$$X(B) = p(E1, B) \cdot L(E1) + p(E2, B) \cdot L(E2) + p(E3, B) \cdot L(E3)$$

$$X(C) = p(E2, C) \cdot L(E2)$$

$$X(D) = p(E2, D) \cdot L(E2) + p(E3, D) \cdot L(E3)$$

and

$$p(E1, B) = 1.0$$

$$p(E2, B) + p(E2, C) + p(E2, D) = 1.0$$

$$p(E3, B) + p(E3, D) = 1.0.$$

The fairness constraint becomes

$$\frac{X(B)}{C_B} = \frac{X(C)}{C_C} = \frac{X(D)}{C_D}.$$

Solving these equations gives new redirection probabilities for E : $p(E1, B) = 1.0$, $p(E2, B) = 0.0$, $p(E2, C) = 0.8795$, $p(E2, D) = 0.1025$, $p(E3, B) = 0.1891$, and $p(E3, D) = 0.8186$.

Since E has failed, the replacement sets for A 's virtual servers are $R(A1) = \{B, C, D\}$, $R(A2) = \{B\}$, $R(A3) = \{B\}$, and $R(A4) = \{B, D\}$. The redirection probabilities for A are calculated using these replacement sets. $\diamond$

Sometimes a system of equations may not be solvable because there is no solution that is perfectly fair. To account for this, we can add error terms (that describe how far we are from satisfying a constraint) to each of the constraints. The error terms catch any load that cannot be distributed fairly. Once modified, we can solve a system of equations as a linear program, with the objective of minimizing the sum of the error terms, to get the fairest solution possible.

4.5 Information Requirements and Solution Times

We mentioned back in Section 4.1 that failure handling should be distributed in nature and fast. We now discuss these two issues as they relate to our approach.

The information needed to calculate redirection probabilities do not conflict with our requirement that failure handling be a distributed process. The information needed by each failure-handling entity includes:

- **The read-sets under the entity’s purview and the virtual servers in those read-sets.** This is local knowledge that should not change much over time. Since there are many redirection entities each of which is responsible for a small number of read-sets, storing this information is a minor burden.
- **The replacement set for each virtual server for which the entity is responsible.** This is global knowledge. However, since replacement sets are derived from the write-sets, which are highly static, the replacement sets do not change often. Thus, each entity can be provided with the replacement sets it needs to know about whenever those sets change.
- **The capacities of each server that may be chosen for redirection.** Since servers enter and leave the system relatively infrequently, making this information available globally is not a problem.
- **The load on each virtual server.** This information can come from either of two sources. If the system is kept perfectly tuned such that it is running as it was designed (as described in [3]), then the load on each virtual server can be computed at system (re-)design time and distributed to all entities. The second source is to have servers monitor how much load is coming from its virtual servers and report that load to the appropriate redirection entities.
- **The failed servers in the union of its replacement sets.** This information can be obtained by pinging servers or through communication timeouts.

As for the speed of calculating redirection probabilities, formulating, but especially solving, a linear program can take a while, even though the solution time is polynomial. We can get around this potential difficulty by pre-calculating some probabilities in advance, for example, for single server failures. Pre-calculating for multiple server failures is not feasible since the number of multiple failure scenarios is exponential. However, we note that some multiple server failure scenarios can be rather common (e.g., servers relying on the same network link), making pre-calculation feasible. In this paper, we do not consider the problem of identifying correlated multiple-failure scenarios.

5 Effectiveness of the Approach

In order to validate our approach, we have conducted experiments for both single- and multiple-server failure scenarios. This section details the results.

To see how fairly load is distributed, we compare replacement servers in terms of **normalized extra load**. Normalized extra load is the percentage of a failed server’s load (or failed servers’ load in case of multiple failures) directed to a replacement server divided by the number of virtual servers the replacement server has. Thus, if load is redistributed fairly, then all the replacement servers in a failure scenario will have the same amount of normalized extra load. For a given failure scenario, the standard deviation of normalized extra load for

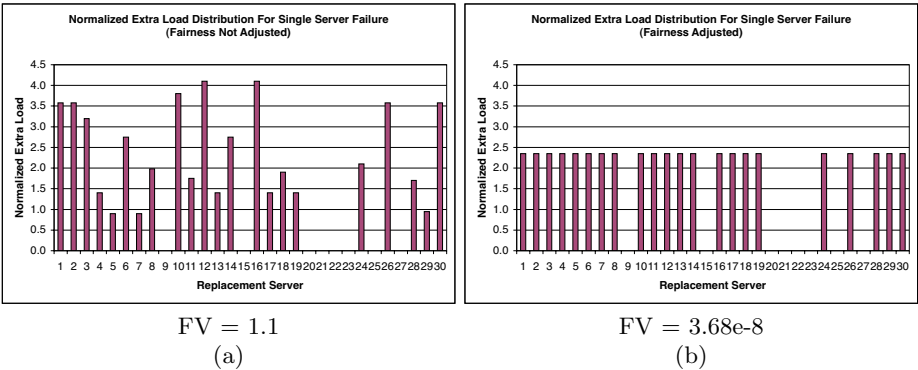


Fig. 4. Example normalized extra load distributions for a single-server failure scenario when (a) fairness is not enforced (b) and when it is enforced.

	fairness not attempted	fairness attempted
mean FV	0.59	0.48
num. scenarios w/ FV $\approx$ 0.0	0	19

Fig. 5. Comparison of fairness values (FVs), for single failures when enforcing fairness was and was not attempted.

the replacement servers, or **fairness value** (FV), indicates how well load was distributed. An extremely fair system has an FV close to 0.0.

In the first set of experiments an 8×8 grid containing 30 servers was used to test the effectiveness of our approach when only one server fails at a time. Each of the 30 servers was independently failed and the loads on the other servers with and without fairly redistributing load was calculated. When fairness was not attempted, replacement servers were selected equiprobably *within the write-set of each failed virtual server*. A typical result is shown in Fig. 4 for the scenario where server 9 fails. When we do not ensure fairness, the servers in 9’s replacement set (all servers except 15, 20-23, 25, and 27) are not fairly loaded as evidenced by the uneven bars of Fig. 4(a). When the fairness algorithm of Fig. 3 is used, a fair load distribution results as the level bars of Fig. 4(b) indicate. In this particular scenario, the unfair system has an FV of 1.1 while the fair system has a value of $3.68\text{e-}8$; the fair system is about 30 million times as fair. Figure 5 compares the FVs of fair systems to those where fairness was not attempted and shows how fairly load was redistributed across all the single-server failure scenarios. While the mean of the FVs in all 30 scenarios are close regardless of whether or not fairness was enforced, the number of scenarios experiencing very low fairness values is much greater (19 versus 0) under fairness.

The second set of experiments focused on multiple server failures. Using the same 8×8 grid as for single failures, groups of two and five servers were picked at random five times each and the servers in a group failed together. Figure 6 shows the normalized load distributions for a 2-failure scenario. Figure 7 contains an

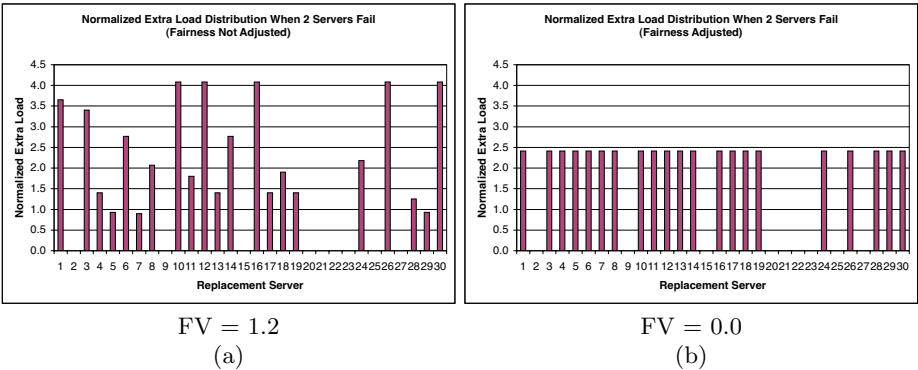


Fig. 6. Normalized extra load distributions when 2 servers have failed when (a) fairness is not adjusted and (b) fairness is adjusted.

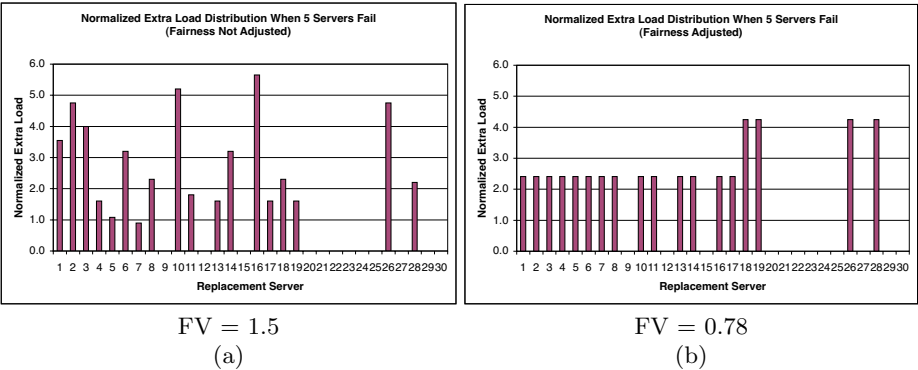


Fig. 7. Normalized extra load distributions when 5 servers have failed when (a) fairness is not adjusted and (b) fairness is adjusted.

	fairness not attempted	fairness attempted
mean FV for the 2-failure scenarios	1.2	0.3
mean FV for the 5-failure scenarios	1.4	1.0

Fig. 8. Comparison of mean standard deviations in server loadings, adjusted for capacity, for 2-failure and 5-failure scenarios.

example 5-failure scenario. In both figures we see that enforcing fairness does a much better job of distributing load than not doing so.

We also see in Fig. 7(b) that perfect fairness was not possible for the 5-failure scenario. This is not surprising since one-sixth of the system’s servers had failed, leaving limited options. Instead, we had to settle for a redistribution that is the fairest possible under the circumstances (see the last paragraph of Section 4.4). Note that extra load was distributed fairly within two groups of servers (the

normalized extra load assumes only two values). This is certainly better than not having attempted fairness at all as the FVs of Figs. 7(a) and 7(b) indicate.

Figure 8 provides a comparison of fairness values for the 2-failure and 5-failure scenarios. Notice that enforcing fairness produces lower average FVs than not doing so. Examining Figs. 5 and 8, we notice that the mean FV rises with the number of failures. Indeed, since redirection options decline as the number of failed servers increases, the odds of finding a fair solution should decline also.

Overall, the experiments indicate our methods for fairly redistributing failed server load works well and has a very good chance of succeeding.

6 Conclusion

We have reviewed our proposed method of replicating large objects such as those found in data grids, data warehouses, and dynamic web sites [3]. Since servers can fail because their read loads become too large, efforts need to be taken to distribute read load fairly to servers so that it is less likely that they overload.

In this paper, we proposed a method to maintain fair server loadings when servers fail. Subject to a set of constraints, we developed a distributed approach for deciding how to re-route requests that would normally have been handled by failed servers to a set of replacement servers. Re-routed requests are distributed fairly to the replacement servers in order to prevent any one server from overloading due to its increased number of requests. We also presented experimental results that verify the validity of our approach.

References

1. J. Kangasharju, J. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *WCW*, 2001.
2. B. Li, M. J. Golin, G. F. Italiano, X. Deng, , and K. Sohrawy. On the optimal placement of web proxies in the internet. In *INFOCOM*, pages 1282–1290, 1999.
3. C. Mayer, K. S. Candan, and V. Sangam. Constraints, parameters, and strategies for replicating large content for web delivery. Technical report, Computer Science and Engineering Department, Arizona State University, 2002.
4. C. B. Mayer, K. S. Candan, and V. Sangam. Effects of user request patterns on a multimedia delivery system. In *Intl Workshop on Multimedia Info Systems*, 2003.
5. L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, 2001.
6. P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed internet replica placement. In *WCW*, 2001.
7. H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. In *HPDC*, pages 76–86, 2001.
8. R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design considerations for distributed caching on the internet. In *ICDCS*, pages 273–284, 1999.
9. U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of e-commerce traffic. In *WECWIS*, pages 137–146, 2002.

Aquarius: A Data-Centric Approach to CORBA Fault-Tolerance

Gregory Chockler, Dahlia Malkhi, Barak Merimovich, and
David Rabinowitz

School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel

{grishac,dalia,barakm,dar}@cs.huji.ac.il

Abstract. In this paper we introduce *Aquarius*, a fault tolerant CORBA software. *Aquarius* seamlessly wraps legacy software using *portable interceptors* to provide a smooth migration path for robustifying existing services. The replication technique it employs is data-centric, which means that a service is considered a shared object, copies of which are stored on independent servers. The shared object is manipulated by clients, without any communication among replicated servers. The data-centric approach differs from other replication systems, in which active replication is implemented by the server replicas.

The Aquarius architecture is demonstrated using a test-case replicated SQL database.

1 Introduction

The Internet provides abundant opportunity to share resources, and form commerce and business relationships. Key to sharing information and performing collaborative tasks are tools that meet client demands for reliability, high availability, and responsiveness. Many techniques for high availability and for load balancing were developed aiming at small to medium clusters. These leave much to be desired when facing highly decentralized settings.

In order to take the existing, successful approaches a step forward towards large scale distributed systems, we identify two core challenge areas related to information technology tools.

The first is attention to scale and dynamism, in order to exploit reliability and survivability techniques in the networks of today and the future. In this domain, there is a growing understanding that replication based on techniques borrowed from the group communication world fail to scale beyond a few dozens of servers, and incur a serious cost of cross-server monitoring for failures. In the past few years, several protocols were developed around quorum replication in a *data-centric* approach, demonstrating both in theory and in practice that it is a viable alternative for fault tolerance. Advances to quorum systems, e.g., [18,21, 3,1,2,14] can be employed according to a wide range of parameters, e.g., to mask server corruption due to malicious penetration, thus offering service developers

flexibility to choose the replication framework most suitable for the application needs. The Fleet project [19,20] and the Agile store [16] are prototype systems demonstrating some of these techniques. Several recent works [11,7,5,6,17] further enhance our fundamental understanding of quorum-based replication. Our work embraces this approach and provides a concrete and detailed implementation and performance assessment.

The second is deployment in real settings and providing an evolution path for legacy software. While we strive to keep the work general, we focus on CORBA [24] as a development platform. This choice is made so as to provide for inter-operability and uniformity, and comply with state-of-the-art heterogeneous middlewares. CORBA is a leading standard for bridging object oriented and distributed systems technologies, simplifying development of distributed applications. Our results provide important insights that may impact the emerging Fault-Tolerant CORBA (FT-CORBA) standard [25].

In this paper we introduce *Aquarius*, a fault tolerant CORBA software that answers the two challenges mentioned above. First, Aquarius employs replication techniques from [7,5] for survivability and scalability. Second, the design seamlessly wraps legacy software to provide a smooth migration path for robustifying existing services. The architecture is demonstrated using a test-case replicated SQL database.

1.1 Data-Centric Replication

Consider a typical service program that is accessible by many clients over the network. The challenge is to robustify the service for high availability and load balancing with little or no intervention to existing client or server code.

The *data-centric* approach regards the service as a shared object which is manipulated by multiple clients. Copies of the object reside on a collection of persistent storage servers, accessed by an unbounded universe of transient client processes. In order to coordinate updates to different copies of an object, clients perform a three-phase commit protocol on the object copies. Their first action is to write their intended update next to the object; then they attempt to commit the update at a quorum; and finally, they commit it and actually invoke the update method on all copies.

The design puts minimal additional functionality on data servers, who neither communicate with one another, nor are aware of each other. Essentially, each server needs a thin wrap that provides a facility for storing and retrieving temporary ‘meta-data’ per object. Clients are also not heavy. Their interaction is through rounds of remote invocations on quorums of servers. The approach offers great simplicity for constructing fault-tolerant distributed systems featuring a high degree of decentralization and scale. In addition, it faithfully models Storage Area Network (SAN) settings where persistent storage elements are directly accessible to clients via a high-speed network.

More concretely, this approach has several important advantages. First, it alleviates the cost of monitoring replicas and reconfiguration upon failures. Second, it provides complete flexibility and autonomy in choosing for each repli-

cated object its replication group, failure threshold, quorum system, and so on. In contrast, most implementations of state machine replication pose a central total-ordering service which is responsible for all replication management (see, e.g., [27,4] for good surveys). Third, it allows support for Byzantine fault tolerance to be easily incorporated into the system by employing Masking Quorum systems [18] and response voting. Finally, limiting redundancy only to the places where it is really needed (namely, object replication) results in an infrastructure with only a few necessary components (cf. Section 3) thus simplifying the system deployment and reducing its code complexity. (Our CORBA implementation uses 4K lines of java code for each of the client and server implementations).

1.2 Fault Tolerance in CORBA

CORBA fault-tolerance has received significant attention in recent years (see e.g., [15,13,23,9,8]), both in research and standardization, culminating with the recently adopted Fault-Tolerant CORBA (FT-CORBA) standard [25]. Traditionally, the existing fault-tolerant CORBA implementations rely on group communication services, such as membership and totally ordered multicast, for supporting consistent object replication. The systems differ mostly at the level at which the group communication support is introduced. Felber classifies in [8] existing systems based on this criterion and identifies three design mainstreams: *integration*, *interception* and *service*. A detailed comparison of these approaches can be found elsewhere (see e.g., [8]).

Two more recent proposals, Interoperable Replication Logic (IRL) [22] and the CORBA fault-tolerance service (FTS) of [10], do not clearly fall in any one of the above categories. IRL [22] is designed according to the three-tier paradigm with the replication logic being implemented by the middle-tier servers. Aquarius borrows the idea of the three tier architecture from IRL. However, in contrast to IRL, the middle tier of Aquarius consists of independent entities that are not aware of each other and do not run any kind of distributed protocol among themselves.

The FTS [10] proposal utilizes the standard CORBA's Portable Object Adaptor (POA) for extending ORB with new features such as fault-tolerance. In particular, FTS introduces a Group Object Adaptor (GOA) which is an extension of POA that provides necessary hooks to support interaction of the standard request processing mechanism and an external group communication system. Aquarius utilizes the object adaptor approach for implementing the server side of the replication support (see Section 4).

1.3 Organization

This paper is structured as follows: Section 2 describes the data-centric methods we employ. Section 3 describes the overall design of *Aquarius*, and details of the implementation are provided in Section 4. Section 5 presents performance measurements of *Aquarius*. Section 6 describes a replicated database built using *Aquarius*. Section 7 outlines possible future developments.

2 Replication Methodology

Our methodology for supporting consistent, universal object replication utilizes the ordering protocol of [5]. The algorithm follows the general Paxos framework to ensure consistent operation ordering at all times (even in presence of failures and timing uncertainty) and guarantee progress when the system stabilizes. Similar to Paxos, the stability assumptions are encapsulated into a separate leader election module.

We assume that a single application object is replicated at $n > 2t$ servers up to t of which can crash. We make use of an RPC communication facility that supports asynchronous invocations and guarantees that an operation issued by a correct client eventually reaches all its correct targets. The algorithm tolerates any number of client failures. We first give a brief description of the ordering algorithm. We then outline the implementation of leader election. Further details and the algorithms and their correctness proofs can be found in [5] and [7], respectively.

2.1 Operation Ordering

The operation ordering is carried out by the client side of the algorithm whose pseudocode appears in the appendix. The client utilizes replicated servers for storing application requests and ordering decisions. The implementation employs two separate threads: one for disseminating application requests to the servers (the *dissemination* thread), and the other one for ordering previously submitted requests (the *ordering* thread). The pseudocode of the dissemination and the ordering threads are depicted in Figures 5 and Figure 6 respectively in the appendix. At the core of the ordering thread is a three-phase Consensus protocol whose decision value is an ordering of operations, represented by a sequence of operation identifiers (**prefix**). The first phase is used to discover the latest decision value which is then extended with newly submitted operations to obtain the next decision value. This new value is then proposed and committed to the servers. The clients employ unique ranks (similar to the Paxos ballots) to prevent concurrent leaders from proposing conflicting decisions and to reliably determine a decision value to extend.

To ensure progress, the ordering thread employs a simple backoff based probabilistic mutual exclusion mechanism similar to that of [7] that works as follows: Whenever an ordering attempt fails because of an intervening ordering attempt by a higher ranked client (PROPOSE returns *nack*), the ordering thread backs off and then repeats its ordering attempt. The backoff period is chosen randomly from the interval $[\Delta, \Delta f(\text{attempt})]$, where Δ is the upper bound on the time required to order a single operation, *attempt* counts the number of unsuccessful ordering attempts, and f is a function monotonically increasing over *attempt* (e.g., $f = 2^{\text{attempt}}$ for exponential backoff). By choosing backoff times from monotonically increasing intervals, the method ensures exclusion among a priori unknown (but eventually bounded) number of simultaneously contending

clients. Note however, that the mechanism does not guarantee starvation freedom if the clients keep submitting new operations. The Aquarius implementation overcomes this problem by introducing persistent client side agents (proxies) and extending the basic backoff protocol to support long-lived leader election.

The server side of the ordering algorithm (see Figure 7 in the appendix) is very simple: It supports only three operations: GET to read the replica's state; PROPOSE to store a possibly non-final ordering proposal whose rank is the highest so far; COMMIT to finalize the order and apply the operations to the application object.

3 System Architecture

3.1 Overview

Our implementation of the data-centric approach forms a middle-tier of client-proxies, whose role is to act on behalf of client requests. Proxy modules may be co-located with client processes, but need not be so. There are several reasons for logically separating between clients and proxies.

Persistence: Proxies can be administered with prudence, and thus a proxy that becomes a leader of the ordering protocol may prevail for a long period. This entails considerable savings in the bootstrap of the ordering scheme. In contrast, clients might enter and leave the system frequently, generating contention for the leadership position.

Efficiency: A proxy may serve multiple clients, as well as multiple objects. Several important optimizations result from this (as described in more detail below), e.g., batching dissemination requests and ordering operations.

Transparency: The client code requires little or no change. For legacy applications this means that they will simply use the remote CORBA reference as they did before, without any changes.

Extendibility: The proxy is an ideal location for extended functionality, as it is in the critical path of the protocol. For example, we envision running in the future monitoring and profiling tools on the distributed application.

On the server side, the data-centric approach requires simple functionality. This simplicity allows us to use the *Object Adapter* approach, introduced in [10]. CORBA defines an object adapter, called the *Portable Object Adapter (POA)*, for use in client-server applications. This adapter can be extended and customized according to the application's specific needs. Aquarius defines the *Quorum Object Adapter (QOA)* which adds the functionality required by the ordering protocol without modifying the application server code.

On the client side, each request is assigned a unique ID used in the ordering protocol. This unique ID is added transparently using the CORBA Portable Interceptor mechanism. This request is sent to the proxy and forwarded to the application servers. No change is required in the client's code.

These then are the main components of the Aquarius system: the QOA that supports the additional server functionality, the proxy that handles dissemination and ordering, and Portable Interceptors that transparently transfer the

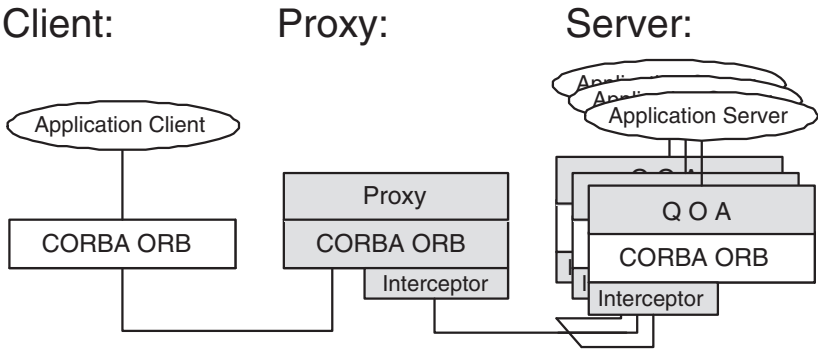


Fig. 1. The *Aquarius* Architecture

unique IDs. This architecture is in the spirit of the data-centric approach, maintaining state on the servers and executing the protocol on the proxy, while enjoying the advantages of a three-tier architecture requiring minimal changes to existing code.

3.2 Proxy

The *Aquarius* proxy is a stateless server - it holds no persistent data, and requires no stable storage. This allows a backup proxy to assume the leadership position in case of a proxy failure without reconfiguration. The new leader automatically acquires all relevant information in the process of executing the ordering protocol.

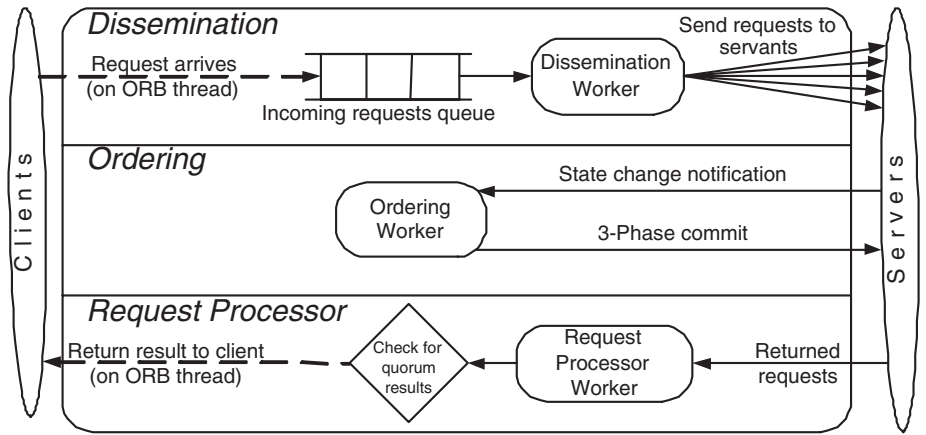


Fig. 2. The *Aquarius* Proxy Architecture

The proxy consists of two parallel threads of execution, each with its own separate data and interface. The first, called the *dissemination thread*, is responsible for disseminating client requests to all replica servers, and to process their responses. The second thread, called the *ordering thread*, is responsible for creating a total order of all client requests. The proxy receives client requests via the CORBA Dynamic Skeleton Interface (DSI) which allows it to receive client calls from any application. The message is then forwarded asynchronously via the CORBA Dynamic Invocation Interface (DII) to all replica servers. Once the results from the replicas return, the proxy returns the agreed upon result to the client.

CORBA Object Request Brokers (ORBs) commonly support two threading paradigms: THREAD-PER-REQUEST and THREAD-POOL. Both these models are inadequate for the tasks required of the Aquarius proxy. Since requests can block until they are ordered a THREAD-POOL may easily be exhausted, halting all future operations in the proxy. The THREAD-PER-REQUEST model is not affected by this problem, but it does not scale well. Therefore a different approach is employed.

The Aquarius proxy uses a fixed number of threads which run fast, non-blocking operations. All incoming requests are queued and dispatched by the dissemination thread. Ordering requests are sent and received by the ordering thread, and a third thread, called the *request processor*, is responsible for collecting replica responses and returning results to the clients. This model uses minimal resources while ensuring the proxy does not halt.

The Aquarius system can manage any number of replicated objects, where each object can have any number of replicas, and be accessed by any number of proxies. For each object, one of the participating proxies is designated as the leader proxy that runs the ordering protocol for that replication group. The other proxies are considered followers, and implicitly rely on the leader to order their requests.

3.3 Quorum Object Adapter (QOA)

In a standard, non fault-tolerant CORBA application, the server object factory instantiates an implementation object and activates it in a CORBA Portable Object Adapter (POA). It then publishes the reference to this object. In *Aquarius*, the object factory must activate the object in a QOA. This is the only change required in the server. Note that the logic of the implementation remains unchanged. Only the factory class, which is usually a separate and much simpler module, is changed.

The QOA acts as a wrapper to the implementation object and an additional CORBA object, which is responsible for handling the server side of the total order protocol. This additional object simply maintains the ordering state of the replica, and changes it according to the ordering calls. It is the simplicity of these calls that allows us to embed them in an object adapter.

Dispatching of client calls in the QOA is handled using the CORBA *RequestDispatcher* mechanism: ordering operations are dispatched by the ordering

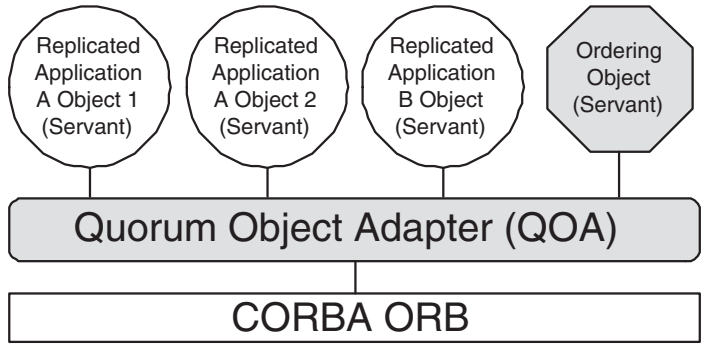


Fig. 3. The *Aquarius* Quorum Object Adapter Architecture

object, and any other operation is kept in storage until they are ready to be executed by the application server object. Operations are ready to be executed once their unique id is committed by the QOA and there is no other unique id prior to it in the total order which has not been executed.

Note that the execution of operations is delegated to a separate thread, so as not to block the ordering object from processing further ordering requests.

4 Implementation

4.1 Bootstrap

An *Aquarius* proxy is responsible for creating or accessing object replicas. These replicas can be created by a call from the proxy, in which case they are created within the context of an *Aquarius* QOA or they can be existing replicas, created previously by an *Aquarius* proxy. The proxy then creates an object group reference, which looks like a standard CORBA reference, and that can be used by any client.

All bootstrap operations use the interfaces defined in the FT-CORBA specification [25]. Specifically, the *GenericFactory* interface is used for creating replicas and groups, and the *PropertyManager* interface is used for specifying configuration parameters.

Once replicas are created and a group reference exists for them, the proxy is responsible for disseminating any requests made on the object group reference to all participating replicas. Any number of proxies can access the object replica-group. Each proxy creates its own object group reference, but all use the same group of replicas. The proxies are independent of each other, and are unaware of any other proxies in the system.

4.2 Ordering

The ordering protocol is a three phase commit protocol, as defined in [7]. The parameters of each phase in the protocol are arrays of unique IDs which detail

the total order of the operations, and pointers to the current position of the total order.

Forming the ordering on message IDs is an optimization of the data-centric approach, as it decouples the dissemination and ordering messages. Each message is assigned a unique ID. The ordering protocol does not require the actual message contents, only its ID, serving to minimize the size of the messages in the ordering protocol. In Aquarius this is supported transparently using CORBA *Portable Interceptors*, which add the unique ID to the message header without any change to the application's client or server code.

An asynchronous notification mechanism between servers and proxies allows the proxy ordering thread to be idle most of the time. As request messages arrive at the replicas, they generate a state change in the server QOA, which notifies one proxy – the current presumed leader – of the change. This initiates the ordering protocol at the proxy leader. Once the message has been committed at the QOA, it is dispatched to the application code, which then executes the request and returns its reply to the proxy that sent it.

In order to support these notifications without registering the proxy at each of the QOAs, a proxy leader calls a remote operation on each of the replicas which only returns if a request is pending at this QOA. If no such operation is pending, the request blocks until a request arrives. This operation is equivalent to the `get()` operation of the ordering protocol, except that it blocks until it can return useful information. The additional remote call adds very little overhead at each of the QOAs while saving the bandwidth required for continuous execution of the ordering protocol.

4.3 Handling Proxy Failures

In order to ensure fault-tolerance all components in the system must be replicated, and the Aquarius proxy is no exception. With multiple proxies in place, a client can overcome a proxy failure by simply switching to any other proxy which is a client for the object, or by instructing a proxy to join this group (this can be implemented transparently with client-side Portable Interceptors, as described in [10]). The situation is more complicated if the proxy that fails is also the leader. In this case, ordering operations stop and the QOAs will not execute any application code. Therefore a new leader must be elected.

A proxy that is part of a group can suspect that the proxy leader for the group has failed if a user-defined timeout has expired since it disseminated a client request to the object replicas. All proxies that suspect such a failure will attempt to become leaders by executing the ordering protocol. Only one will succeed, and it becomes the new leader, while the others will fail on a *RankException* and remain followers. The statelessness of the ordering protocol makes this possible, since a new leader can simply resume where the last one failed.

4.4 Summary of Optimizations and Enhancements

While implementing the system we found some practical improvements of value: **Garbage collection:** Maintaining and transmitting the state of the ordering protocol over time requires a growing amount of resources. *Aquarius* minimizes this by adding the notion of a 'stable line'. This is the most recent command that has been executed by *all* servers. The state before this request can be discarded. This saves considerable amounts of information that must be sent between the proxy and the QOAs.

Message batching: The proxy can batch the ordering of multiple messages, sent by multiple clients in one ordering message, thus saving the bandwidth and round trip time required for the remote calls.

Notifications: The notifications described above are a practical solution required for the proxy architecture, which also increases the efficiency of the system.

Threading model: The specially designed threading model described above requires a constant amount of memory allowing for greater scalability, while ensuring that the proxy never blocks.

5 Performance

This section outlines measurements for the *Aquarius* system in a test environment. The system was implemented using ORBacus 4.1.0 [26] and the Java language, using JDK 1.3.1. The experiments were performed on Pentium III PCs over a 100Mbps local area network. Each PC is equipped with a 500Mhz CPU, 256MB of RAM, and runs the Debian Linux (kernel 2.4.18) operating system.

A simple client/server was developed for the experiments. The server contains a single remote operation that receives a buffer of varying size as a parameter. This allows us to measure the round trip time of a request as affected by the size of the request. Two sets of tests were run: the first tests the performance of the system with one client and proxy, and an increasing number of application servers (equivalently, the replication degree). The second test increases the number of concurrent clients while working with a single proxy and five application servers. The results of both tests are depicted in Figure 4.

6 A Database Application

Recent years witnessed a great interest in replicated systems, databases in particular. The need to maintain the availability of commercial data led to the development of several database replication techniques. All major database vendors support some sort of replication for their products. Other companies offer middleware that enables replication.

Our approach offers an easy-to-construct replication middleware. The combination of such a middleware with a standard, non-replicated, database is a cheap alternative to commercial products.

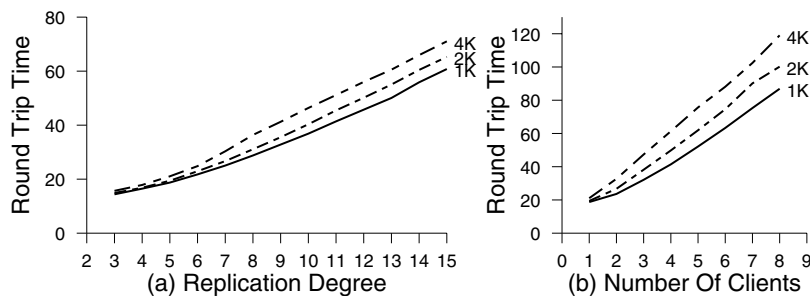


Fig. 4. Round trip time (in milliseconds): (a) according to replication degree and request size; (b) according to number of concurrent clients and request size (Using one proxy and 5 application servers)

We have built a prototype replicated database using our methods over the HSQL [12] database, an open source relational database system written in java. HSQL is a JDBC-compliant SQL database, but has no replication support. By combining HSQL with the Aquarius system, replication is achieved with very little additional code. A simple server object was written in java, that supports a single remote operation. This operation receives an SQL query, and returns its result. The server itself requires only two hundred lines of code, including server initialization and configuration. The resulting system shows good scalability, and supports 50 operations per second for 5 replicas on the test environment described above.

7 Future Directions

Future work on the *Aquarius* system includes several possible directions.

Quorum definitions: The pluggable quorum management module allows new quorum systems to be tested for efficiency. In addition, the module responsible for communicating with quorums may be extended beyond strict quorums access. Two possible examples is allowing asynchronous backup of a slower secondary site (without slowing the primary site) or supporting dirty reads (read operations that do not require a quorum of replies, risking reading old information).

Recovery: Transferring the state of an object replica-group to a new replica, or to a faulty one that recovers.

Monitoring and Security: Proxies are an excellent location for handling system monitoring and maintaining access control lists. Portable Interceptors can be used to add these features transparently to the system.

References

1. L. Alvisi, J.P. Martin and M. Dahlin. Minimal Byzantine Storage. In *Proceedings of the 16th International Symposium on Distributed Computing (DISC 2002)*, Toulouse, France, October 2002, pp. 311–326.
2. L. Alvisi, J.P. Martin and M. Dahlin. Small Byzantine Quorum Systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002 and FTCS 32)*, DCC Symposium, Washington, DC, June 2002, pp. 374–383.
3. R. Bazzi. Synchronous Byzantine Quorum Systems. *Distributed Computing* 13(1), pages 45–52, 2000.
4. G. V. Chockler, I. Keidar and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys* 33(4): 1–43, December 2001.
5. G. Chockler and D. Malkhi. Active Disk Paxos with infinitely many processes. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, July 20–24, 2002, Monterey, California, USA.
6. G. Chokler, D. Malkhi and D. Dolev. A data-centric approach for scalable state machine replication. In *Future Directions in Distributed Computing, Lecture Notes in Computer Science Volume 2584*, Springer-Verlag, 2003.
7. G. Chockler, D. Malkhi, and M. K. Reiter. Backoff protocols for distributed mutual exclusion and ordering. *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 11–20, April 2001.
8. P. Felber. The CORBA Object Group Service. A service approach to object groups in CORBA. *PhD Thesis, Ecole Polytechnique Federale de Lausanne*, 1998.
9. P. Felber and R. Guerraoui and A. Schiper. The implementation of a CORBA object group service. *Theory and Practice of Object Systems*, 4(2): 93–105, 1998.
10. R. Friedman and E. Hadad. FTS: A High-Performance CORBA Fault-Tolerance Service. In *The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*
11. E. Gafni and L. Lamport. Disk Paxos. *Proceedings of 14th International Symposium on Distributed Computing (DISC'2000)*, pages 330–344, October 2000.
12. HSQL Database
<http://www.hsqldb.org>
13. IONA and Isis. An Introduction to Orbix+ISIS. *IONA Technologies Ltd. and Isis Distributed Systems, Inc.*, 1994.
14. S. Lakshmanan, M. Ahamad, H. Venkateswaran. A secure and highly available distributed store for meeting diverse data storage needs. In *proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, July 2001, Goteborg, Sweden.
15. S. Landis and S. Maffei. Building reliable distributed systems with CORBA. *Theory and Practice of Object Systems*, 3(1), 1997.
16. S. Lakshmanan, M. Ahamad and H. Venkateswaran. Responsive security for stored data, In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2003)*.
17. D. Malkhi. From Byzantine Agreement to Practical Survivability; A position paper. In *Proceedings of the International Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS 2002)*, October 2002, Osaka, Japan.
18. D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing* 11(4): 203–213, 1998.

19. D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering* 12(2): 187–202, March/April 2000.
20. D. Malkhi, M. K. Reiter, D. Tulone and E. Ziskind. Persistent Objects in the Fleet System. In *DARPA's second DARPA Information Survivability Conference and Exposition (DISCEX II)*, California, June 2001.
21. D. Malkhi, M. Reiter and A. Wool. The load and availability of Byzantine quorum systems. *SIAM Journal of Computing* 29(6): 1889–1906, 2000.
22. C. Marchetti, A. Virgillito, R. Baldoni. Design of an Interoperable FT-CORBA Compliant Infrastructure. In *proceedings of the 4th European Research Seminar on Advances in Distributed Systems Dependable Systems (ERSADS'01)*.
23. L. E. Moser and P. M. Meliar-Smith and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2): 81–92, 1998.
24. Object Management Group. The Common Object Request Broker: Architecture and Specification, 2.3 edition, June 1999.
25. Object Management Group. Fault Tolerant CORBA Specification, *OMG Document ptc/2000-04-04*, April 2000.
26. IONA's ORBacus http://www.iona.com/products/orbacus_home.htm
27. D. Powell, editor. Group communication. *Communications of the ACM* 39(4), April 1996.

A Pseudo Code of the Ordering Protocol

```

Boolean finish = false;
Dissemination thread:
    When an operation op is submitted for ordering:
        Assign op a unique id;
        Invoke SUBMIT(id, op) on all servers;
        Wait until some server responds with  $\langle id, res \rangle$ ;
        finish  $\leftarrow$  true;
        Return res;

```

Fig. 5. Data-centric operation ordering: The client request dissemination thread

Ordering thread:

```

do
  Wait until (isLeader  $\vee$  finish);
  While (isLeader  $\wedge$   $\neg$ finish) do
    Pick a unique, monotonically increasing rank r;
    Invoke GET(r) on n servers;
    Wait for more than  $n/2$  servers si to respond with  $\langle r_i, \text{prefix}_i, \text{pending}_i \rangle$ ;
    Let Pending =  $\bigcup_i \text{pending}_i$ ;
    Let prefix = prefixj such that  $r_j = \max_i r_i$ ;
    For each id  $\in$  Pending which is not included in prefix
      prefix  $\leftarrow$  append(prefix, id);
    Invoke PROPOSE(r, prefix) at all servers;
    Wait for more than  $n/2$  servers si to respond with ack/abort;
    If more than  $n/2$  servers respond with ack
      Invoke COMMIT(r, prefix) on all servers;
  od
Until (finish)

```

Fig. 6. Data-centric operation ordering: The client ordering thread

Sets *pending*, *Ops*, initially empty;

Sequences *prefix^p*, *prefix^c*, initially empty;

Ranks *getRank*, *propRank*, initialized to a predefined initial value;

SUBMIT(<i>id</i>, <i>op</i>): <i>pending</i> $\leftarrow$ <i>pending</i> $\cup$ {<i>id</i>}; <i>Ops</i> $\leftarrow$ <i>Ops</i> $\cup$ {<i>id</i>, <i>op</i>}; Execute WAITANDAPPLY(<i>id</i>) in a separate thread; GET(<i>r</i>): if (<i>r</i> > <i>getRank</i>) <i>getRank</i> $\leftarrow$ <i>r</i>; return $\langle \text{propRank}, \text{prefix}^p, \text{pending} \rangle$; PROPOSE(<i>r</i>, <i>prefix</i>): if (<i>getRank</i> $\leq$ <i>r</i> $\vee$ <i>propRank</i> < <i>r</i>) <i>propRank</i> $\leftarrow$ <i>r</i>; <i>prefix^p</i> $\leftarrow$ <i>prefix</i>; return <i>ack</i>; return <i>nack</i>;	WAITANDAPPLY(<i>id</i>): Wait until: (1) <i>id</i> appears on <i>prefix^c</i>; (2) all operations preceding <i>id</i> in <i>prefix^c</i> were applied; (3) $\langle id, op \rangle \in \text{Ops}$ for some operation <i>op</i>; Apply <i>op</i> to the application object and return the result to client; COMMIT(<i>r</i>, <i>prefix</i>): if (<i>propRank</i> $\leq$ <i>r</i>) <i>prefix^c</i> $\leftarrow$ <i>prefix</i>; return <i>ack</i>;

Fig. 7. Data-centric operation ordering: The server side

FT-JAVA: A Java-Based Framework for Fault-Tolerant Distributed Software

Vicraj Thomas, Andrew McMullen, and Lee Graba

Honeywell Laboratories, Minneapolis MN 55418, USA

Vic.Thomas@Honeywell.com

Abstract. FT-JAVA is a Java language based framework for building fault-tolerant distributed software. It is designed to bring to the system programmer much of the flexibility provided by reflective languages and systems without the attendant difficulties of reasoning about correct system structure. FT-JAVA achieves this by providing the programmer an extremely flexible programming model with sufficient structure to enable reasoning about the system. The FT-JAVA framework in turn uses the power of reflection to implement the programming model.

1 Introduction

An increasing number of mission-critical distributed systems are being developed using the Java programming language. Examples range from aerospace applications such as portions of the software on NASA's Mars Pathfinder and Odyssey missions to financial applications such as the Den norske Bank's Internet banking facility. Its use in critical applications will only increase as implementations of the Real-Time Java Specification [3] become available.

FT-JAVA is a Java language framework for building *fault-tolerant distributed software*—software that must continue to provide service in a multicomputer system despite failures in the underlying computing platform. The framework supports a variety of fault-tolerance design patterns (software structuring techniques) because Java is used to build a variety of different types of systems; different patterns are appropriate for different types of systems or even different parts of the same system.

FT-JAVA recognizes the expressive power of *reflection* [4] in building systems that support a variety of fault-tolerance design patterns. FT-JAVA also recognizes reflection can be a double-edged sword: the same flexibility that allows for the implementation of different system structures also allows for the system to be structured in many incorrect ways. This flexibility also makes it very difficult to reason about the runtime behaviour of such systems.

FT-JAVA provides system developers with a framework that reaps the benefits of reflection and yet imposes sufficient structure to permit reasoning about program behaviour. This is achieved by means of a programming model that shields developers from the complexities of reflection while still being flexible

enough to use a variety of fault-tolerance design patterns. The FT-JAVA framework itself uses reflection to provide this flexible programming model but this reflection is not visible to system developers.

FT-JAVA supports the *fail-stop modules* programming model [12]. This model supports different fault-tolerance design patterns including replicated state machines [13] and restartable actions [11]. FT-JAVA uses Java reflection to implement this programming model. It does not require any extensions to the Java language or runtime and is therefore easily deployed on any system with a standard Java virtual machine.

2 The Fail-Stop Modules Programming Model

The fail-stop failure model for processors [11] has proven to be a very convenient abstraction for analyzing the behaviour of distributed systems in the face of processor failures. This model assumes processors produce correct results until they fail and the failure of a processor is detectable by other functioning components.

The fail-stop modules programming model extends this abstraction to software modules. A *fail-stop module* (or FS module) is an abstract unit of encapsulation. It implements a collection of operations that may be invoked by other FS modules. When an operation is invoked it executes to completion as an atomic unit, despite failures and concurrent invocations. The failure resiliency of an FS module is increased either by composing modules to form complex FS modules, or by using recovery techniques within the module itself. Replicating a module N times on separate processors to create a high-level abstract module that can survive $N - 1$ failures is an example of the former, while including a recovery protocol that reads checkpointed state is an example of the latter.

A key aspect of FS modules is *failure notification*. Notification is generated whenever a failure exhausts the redundancy of a FS module, resulting in a failure of the abstraction being implemented. The notification can be fielded by other modules that use the failed module so they can react to the loss of functionality.

The failure notification and composability aspects of FS modules are what makes this programming model so useful. Ideally, a fault-tolerant system as a whole behaves as an FS module: commands to the program are executed atomically or a failure notification is generated. This assures users that, unless a failure notification is generated, their commands have been correctly processed and any results produced can be relied upon. Such a program is much easier to design and implement if each of its components are in turn implemented as FS modules. Since the failure of any component is detectable, other components do not have to implement complicated failure detection schemes or deal with the possibility of erroneous results produced by failed components. These components may in turn be implemented by other FS modules, and this process continued until the simplest components are implemented by *simple FS modules*. At each level, the guarantees made by FS modules simplify the composition process [12].

The fail-stop module programming model does not assume fail-stop semantics for the underlying computation platform. The responsibility of detecting

platform failures is delegated to failure detectors. Different types of failure detectors are used depending on the failure model assumed for the computation platform.

3 FT-JAVA Overview

In the FT-JAVA framework FS modules are Java classes that extend an abstract class `FSObject`. Operations on FS modules are implemented as methods on one or more `FSInterface` interfaces. The atomicity of execution of operations may be ensured by using Java's `synchronized` modifier on methods of the `FSInterface` interfaces.

Complex FS modules with greater failure resilience are formed by composing other FS modules. In addition to standard Java composition techniques such as object containment and delegation, FT-JAVA allows FS modules to be composed by replication. Multiple instances of an FS module may be composed as a group with FT-JAVA providing the illusion the group is a single FS module. Operations invoked on the group are multicast to all members of the group and failure notifications are generated when all group members have failed. Finally, complex FS modules may also be implemented using the *restartable actions* pattern [11] where failed modules are restarted from checkpointed state.

Two kinds of failure notifications may be generated when an FS module fails: (1) a synchronous notification, generated when a method is invoked on the failed FS module and (2) an asynchronous notification that is generated even if no method is being invoked on the failed module. Synchronous failure notifications are implemented as Java exceptions—all methods on `FSInterfaces` throw an exception that must be caught or propagated by callers of these methods. Asynchronous failure notifications follow the “event subscriber” model used by Java Swing components. Modules ask to be notified of the failure of an FS module by invoking its `addFailureListener` method. This method takes as an argument a `FailureListener` interface; the `failureEvent` method on this interface is invoked when the FS module fails.

4 Programming with FT-JAVA

4.1 Defining and Creating FS Modules

All FS modules in FT-JAVA inherit from an abstract class `FSObject`. An FS module's class is not expected to implement any methods defined by the base class `FSObject`. However, the constructor of the FS module's class is expected to make a stylized call to the base class' constructor. This is illustrated in Fig. 1 that shows a simple FS module.

FS modules are created by invoking the `newInstance` method of the FT-JAVA class `FSObject`. The following statement demonstrates the creation of an instance of the `BankAccountImpl` FS module on a host named Caesar.

```

class BankAccountImpl extends FSObject implements BankAccount {
    private int currentBalance = 0;

    /* class constructor */
    public BankAccountImpl(ActivationID actID,
                           MarshalledObject args) throws RemoteException {
        super(actID, args);
    }

    /* method from interface BankAccount */
    public synchronized int makeDeposit(int deposit)
        throws RemoteException {
        currentBalance += deposit; return currentBalance;
    }

    /* implementations of other methods */
    ...
}

interface BankAccount extends FSInterface {
    public synchronized int makeDeposit(int deposit)
        throws RemoteException;
    ...
}

```

Fig. 1. A simple FS module

```

BankAccount custAccount =
    (BankAccount)FSObject.newInstance("BankAccountImpl",
        new String[] {"Caesar"});

```

The first argument to `newInstance` is the name of class of the FS module and the second argument is an array with the name of the host on which the FS module is to be created. Of course, the name of the host would typically not be programmed into the application but would instead be obtained from a configuration application, file or database.

The `newInstance` method is also used to create a new FS module by composing other FS modules by replication. The following statement shows the creation of a more failure resilient version of the `BankAccountImpl` FS module by creating instances on two different hosts.

```

BankAccount custAccount =
    (BankAccount)FSObject.newInstance("BankAccountImpl",
        new String[] {"Caesar", "Czar"});

```

The first argument is the class of the FS module and the second argument is an array of names of hosts on which instances are to be created. Note the reference to the replicated group returned by the `newInstance` method is indistinguishable from a reference to a single object instance.

4.2 Invoking Methods on FS Modules

A method invocation on an FS module is no different from a method invocation on a regular Java object. This is true regardless of whether the FS module is replicated or not. The following code fragment shows the invocation of the method `makeDeposit` on the `BankAccount` FS module created in the previous section.

```
try {
    custAccount.makeDeposit(100);
}
catch (RemoteException e) {
    System.err.println("Invocation failed...");
}
```

Note that all methods on FS modules can throw a `RemoteException`. This exception must therefore be caught or propagated.

4.3 Failure Notification

Asynchronous notification of the failure of an FS module is obtained by registering a `failureListener` for the module by invoking its `addFailureListener` method. Declaration and implementation of the `addFailureListener` method is transparent to the FS module developer as it is defined on an interface implemented by the `FSObject` base class. The following code fragment illustrates the creation and registration of a failure listener for the FS module used in previous examples.

```
custAccount.addFailureListener(new FailureListener {
    public void failureEvent() {
        System.out.println("BankAccount object failed.");
    }
})
```

In the case of an FS module composed by replication, failure notifications are generated only when all instances have failed.

A synchronous failure notification is generated when a method invocation on an FS module fails or if a method is invoked on a failed FS module. This failure notification is a Java exception that must be caught and handled using the Java `try/catch` statement, as shown in Sect. 4.2.

5 FT-JAVA Implementation

5.1 Overview

FT-JAVA is implemented using version 1.4.1 of the Java Platform Standard Edition. It is implemented as a `faultTolerance` Java package.

The implementation of FT-JAVA makes extensive use of the Java Reflection API [2] and the Java Remote Method Invocation (RMI) package [6]. The Java Reflection API includes a `Proxy` class that provides methods for creating *dynamic proxy* classes and instances. A dynamic proxy implements a list of interfaces specified when the proxy instance is created. Associated with the proxy instance is an *invocation handler object*. Invocations on the dynamic proxy are dispatched to the `invoke` method of the invocation handler object. This handler processes the method invocation as appropriate and the result it returns is returned in turn as the result of the method invocation on the dynamic proxy instance [14]. As described later in this section, all invocations on FT-JAVA FS modules go through a dynamic proxy that is co-located (in the same Java virtual machine) with the caller.

The Java RMI package is used to support distributed object creation and invocation. The package defines an `Activatable` class whose subtypes can be created remotely and activated on demand. It also defines a `Remote` interface; all remotely invocable interfaces must inherit from this interface. As described later in this section, all FS modules are a sub-type of the `Activatable` class and all `FSInterfaces` inherit from the `Remote` interface.

The FT-JAVA architecture is shown in Fig. 2. Three services run on all hosts of a FT-JAVA system: the Java RMI Registry, the Java RMI Daemon, and a fault-tolerance manager. The RMI Registry and RMI Daemon are part of the standard Java platform; the former is the Java name service and the latter supports the creation of new Java virtual machines (JVM) at runtime and the creation of Java objects in JVMs on the request of objects in other JVMs.

The fault-tolerance manager service, called the `HostFTManager`, is responsible for managing the Java virtual machines on the host. It uses the Java RMI Daemon to create new virtual machines and monitors these virtual machines for failure. Associated with the `HostFTManager` service is a detector for failure of other hosts in the system. The failure detector to be used is specified as a system parameter when the `HostFTManager` service is started up.

FT-JAVA allows multiple *applications* to co-exist. An application is a collection of related Java objects. On a given host an application is a process consisting of a JVM and all the objects on that host that belong to that application. By default, an FS module is created in an application of the same name as the creator of the FS module. This can be overridden by a parameter on the `FSObject.newInstance` method used to create FS modules.

5.2 FS Module Creation

The `newInstance` method on `FSObject` used to create FS modules takes as arguments the Java class name for the FS module, a list of hosts on which

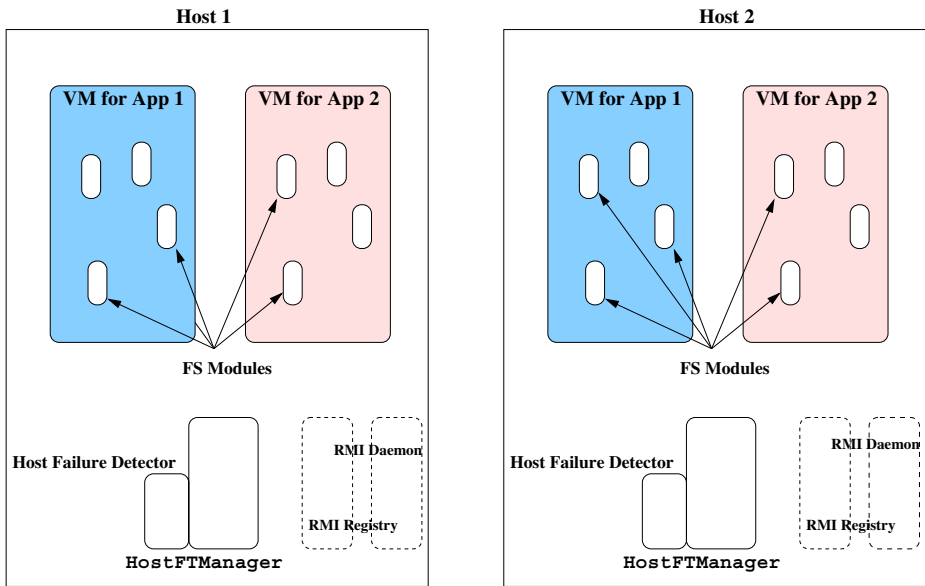


Fig. 2. FT-JAVA Architecture

instances of the FS module are to be created, a list of backup hosts on which new instances are to be created if an active instance fails, and the name of the application that owns the FS module. Not all of these parameter values have to be specified—reasonable defaults are used for unspecified parameter values.

The following are the major steps executed by `FSObject.newInstance`.

1. For each host on which an instance of the FS module is to be created:
 - a) Use the Java name service (RMI Registry) to find the `HostFTManager` on the host.
 - b) Invoke the `createObject` method on the `HostFTManager` with the application name and class name of the FS module as parameters. The `HostFTManager.createObject()` method on the host on which the FS module is to be created executes the following major steps:
 - i. Determine if a JVM already exists for the application on that host.
 - ii. If a JVM does not already exist, create one using the activation services provided by the Java RMI daemon.
 - iii. Activate an instance of the FS module class and return a reference to the instance. (The instance will actually be instantiated on the first invocation on the object.)
 - c) Save reference to instance returned by `HostFTManager.createObject`
2. Use the Java reflection API to get a list of all FS interfaces (interfaces of type `FSInterface`) implemented by the FS module.
3. Create a dynamic proxy with the above FS interfaces.
4. Return reference to the proxy.

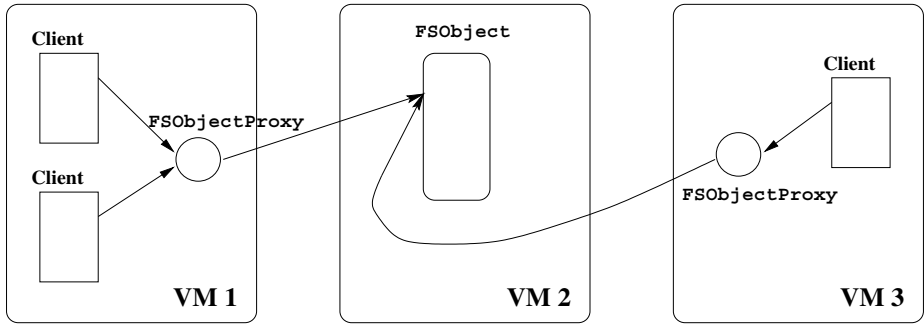


Fig. 3. Method invocations on an **FSObject** intercepted by **FSObjectProxy** objects

5.3 Invocations on FS Modules

A method invocation on an FS module is essentially a method invocation on a local proxy to the FS module. The proxy class delegates all invocations to a handler called **FSObjectProxy**. The handler maintains a list of references to all object instances that comprise the FS module. The handler invokes the appropriate method on all of the object instances and returns to the caller the value returned by one of the invocations. Figure 3 shows the invocation of a method on an **FSObject** instance via **FSObjectProxy** instances.

5.4 Registering for Failure Notifications

Any object can ask to be notified of the failure of an FS module by invoking the **addFailureListener** method on the FS module. This method is defined on a **FailureNotification** interface. The **FSObject** class implements this interface and since all FS modules inherit from **FSObject** they automatically implement the **FailureNotification** interface. The **addFailureListener** method takes as its argument a **FailureListener** interface. The **failureEvent** method on this interface is invoked on the failure of the FS module for which it is a listener.

Invocations of **addFailureListener** method on an FS module are handled differently from invocations of other methods on the FS module. As with other invocations, this method invocation is delegated by the module's proxy to its **FSObjectProxy** handler object. The handler notices the **addFailureListener** method is being invoked and performs the following steps:

1. For each instance of this FS module:
 - a) Create an **InstanceFailureHandler** object to handle failures of this instance.
 - b) Ask the local **HostFTManager** for notification when the remote host on which the instance exists fails by invoking its **addHostFailureListener** method with the **InstanceFailureHandler** object and the remote host name as arguments.

- c) Use the Java name service (RMI Registry) to find the **HostFTManager** of the remote host on which this instance of the FS module lives.
- d) Ask the remote **HostFTManager** for notification of the failure of the application (JVM) of the instance by invoking its **addVMFailureListener** method with the **InstanceFailureHandler** object and application name as arguments.

5.5 Failure Detection and Notification

Failure Detection. FT-JAVA uses two levels of protocols to detect the failure of FS modules: one to detect failures of Java virtual machines and another to detect failures of hosts. Each **HostFTManager** is responsible for detecting the failure of virtual machines for applications on its host. A simple heartbeat protocol between the **HostFTManager** and an **AppFTManager** in each of the virtual machines is used to detect these failures.

A second protocol is used to detect the failure of hosts in the system. Associated with each **HostFTManager** is a **HostFailureDetector** responsible for monitoring the health of other hosts in the system. FT-JAVA does not dictate a host failure detector protocol: it is designed to accommodate different failure detectors depending on the failure semantics of the host compute platform. The class name of the specific failure detector to be used is specified as a system parameter; this class must be of type **HostFailureDetector**. A simple heartbeat based protocol that assumes a crash failure model for hosts is distributed with FT-JAVA.

Failure Notification. An FS module instance fails when the virtual machine on which it is executing fails or when the host on which it is executing fails. When a **HostFTManager** detects the failure of a virtual machine on its host, it invokes the **failureEvent** method on all of the registered **InstanceFailureHandlers** for all FS modules on the failed virtual machine. Recall these failure handlers were registered by proxies for these FS modules using the **addVMFailureListener** method of the **HostFTManager**. These proxies may be on any host in the system.

When a **HostFTManager** is informed of the failure of a host in the system, it invokes the **failureEvent** method on all registered **InstanceFailureHandlers** for all FS modules on the failed host. Recall these failure handlers were registered by proxies local to this **HostFTManager** by invoking its **addHostFailureListener** method.

A client that asked to be informed when an FS module fails is notified only when all instances of the FS module have failed. When an instance of an FS module fails, **InstanceFailureHandlers** for the instance are notified as described above. The **InstanceFailureHandlers** simply mark the instance as having failed. The **failureEvent** methods of the **FailureListeners** for the FS module are invoked only when all instances of the FS module are marked as having failed.

A synchronous failure notification (Java exception) is raised when a method is invoked on a failed FS module i.e. all instances of the FS module are marked as having failed.

5.6 Managing State Consistency across Replicated FS-Modules

A future version of FT-JAVA will include better support for managing state consistency across replicated objects. We have designed a scheme that enables the easy insertion of any group message ordering protocol to manage state consistency. In this scheme, shown in Fig. 4, message ordering protocols are implemented as dynamic proxies (**GroupProxy** objects) that front instances of a replicated FS module. An invocation on a FS module instance is trapped by its **GroupProxy**. The proxy communicates with other corresponding **GroupProxy** instances to agree on a consistent message order. All **GroupProxy** instances pass invocations on to their corresponding FS module instances in the agreed upon order. Note that the message ordering protocol is completely transparent to the FS module being replicated and to the clients that invoke methods on the FS module.

5.7 Performance

The overhead associated with invoking a method on an FS module via the dynamic proxy over a direct invocation on a remote object was found to be less than one millisecond, the granularity of Java’s standard timing mechanism. The

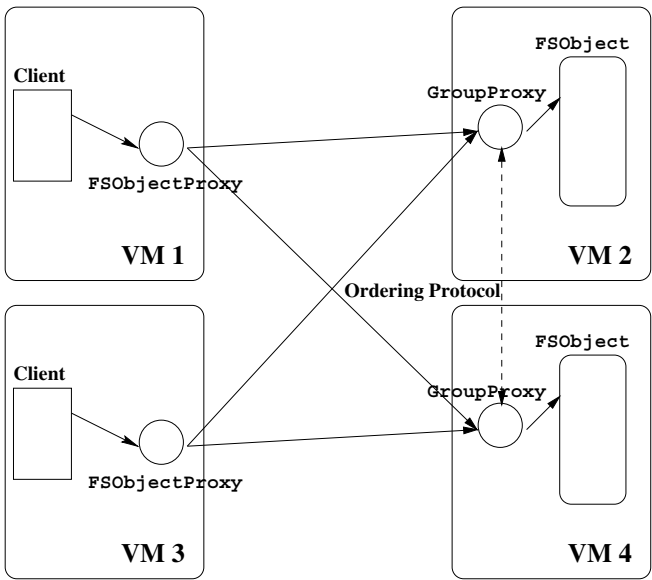


Fig. 4. Scheme for implementing message ordering protocols

time to directly invoke a method on a remote FS module was 3 milliseconds. Going through the proxy did not measurably increase this invocation time.

The above times were measured by having a client repeatedly invoke a method on an FS module on a different node. The nodes were 1.2GHz Pentium III class machines running Windows XP and connected by an 11Mbps wireless Ethernet network.

6 Related Work

Many research projects have explored the use of reflection for building fault-tolerant systems. These projects define *meta-objects* that reify dependability related properties of the functional objects (or *base objects*) of the system. They also define a *meta-object protocol (MOP)* for manipulating these meta-objects to modify the runtime behaviour of corresponding base objects. Maud [1] uses reflection to allow a running system to dynamically switch between different dependability modes of execution by installing and removing dependability protocols. The FRIENDS project [5] showed how MOPs may be used to implement a number of fault-tolerant mechanisms including replication, synchronization, and voting. A MOP for a fault-tolerant CORBA system is described in [8].

In contrast with the above reflection based systems, FT-JAVA shields the system developer from the complexities of MOPs. In fact, the developer does not have to know anything about reflection and MOPs to use the FT-JAVA framework. The programming model seen by the developer is the fail-stop modules programming model; reflection allows for a simple and elegant implementation of support for this model within FT-JAVA.

Systems that provide mechanisms for creating highly-available modules by replication include the Object Management Group's Fault-Tolerance Specification for CORBA [7] and the Aroma System [10]. The fault-tolerance specification for CORBA defines a programming language independent mechanism for deploying highly-available applications. It provides mechanisms for replicating CORBA servers but, unlike FT-JAVA, it does not define a general programming model for building highly-available applications. The fail-stop module programming model of FT-JAVA is such a model with replication being just one technique for building fault-tolerant software. In addition, the FT-JAVA framework helps system designers reason about the structure and behaviour of the system and requires them to deal with both anticipated and unanticipated failures of system modules.

The Aroma System [10] provides mechanisms for creating and managing replicated Java RMI objects. It differs from FT-JAVA in two significant ways: (1) it is implemented as a software layer embedded within the transport layer of the Java RMI stack, and (2) it does not provide system designers a framework that supports fault-tolerance techniques other than replication. In contrast, FT-JAVA requires no extensions to the Java language or virtual machine and supports programming paradigms such as restartable actions in addition to replication.

7 Summary and Future Work

FT-JAVA provides developers of Java-based fault-tolerant software systems a structured way of harnessing much of the power of reflective systems. Its mechanisms integrate well with existing Java constructs, making them easy to learn and use. FT-JAVA does not make any extensions to the Java language or runtime.

Much of the implementation of FT-JAVA is complete and is available as a Java package called `faultTolerance`. Two major FT-JAVA features that have been designed but have yet to be implemented are the restart of failed FS modules on backup hosts and the detection of the failure of an application (JVM) on a host.

Future versions of FT-JAVA will include better support for managing state consistency across replicated FS module as described in Sect. 5.6. A related direction for this work is support for other replication models such as passive replication.

An orthogonal direction is the verification of our hypothesis that the fail-stop failure model seen by the programmer can be maintained even if the host failure model changes from crash failures to Byzantine failures [9]. Naturally, a different host failure detector protocol will be needed for different failure models but this is hidden from the programmer.

References

1. G. Agha, S. Frølund, R. Panwar, and D. Sturman. A linguistic framework for dynamic composition of dependability protocols. In *Proceedings of the IFIP Conference on Dependable Computing for Critical Applications*, Sicily, 1992.
2. K. Arnold, J. Gosling, and D. Holmes. *The JavaTM Programming Language, Third Edition*, chapter 11. Addison-Wesley Publishing Company, June 2000.
3. G. Bollela, J. Gosling, B. Brosgol, P. Dibble, S. Furr, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley Publishing Company, January 2000.
4. F.-N. Demers and J. Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI*, pages 29–38, August 1995.
5. J.-C. Fabre and T. Perennou. A metaobject architecture for fault-tolerant distributed systems: The FRIENDS approach. *IEEE Transactions on Computers*, 47(1): 78–95, 1998.
6. W. Grosso. *Java RMI*. O'Reilly and Associates, October 2001.
7. Object Management Group. *Common Object Request Broker Architecture: Core Specification*, chapter 23. Object Management Group (OMG), December 2002.
8. M.-O. Killijian and J.-C. Fabre. Implementing a reflective fault-tolerant CORBA system. In *Symposium on Reliable Distributed Systems*, pages 154–163, 2000.
9. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3): 382–401, July 1982.
10. N. Narasimhan, L. Moser, and P. Melliar-Smith. Transparent consistent replication of Java RMI objects. In *2nd International Symposium on Distributed Objects and Applications (DOA 2000)*, pages 17–26, September 2000.

11. R. Schlichting and F. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *IEEE Transactions on Computing Systems*, 1(3): 222–238, August 1983.
12. R. Schlichting and V. Thomas. Programming language support for writing fault-tolerant distributed systems. *IEEE Transactions on Computers*, 44(2): 203–212, February 1995.
13. F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4): 299–319, December 1990.
14. Sun Microsystems. *JavaTM 2 Platform, Standard Edition, v 1.4.1, API Specification*, 2002. <http://java.sun.com/apis.html>.

Adaptive Message Packing for Group Communication Systems^{*}

Alberto Bartoli¹, Cosimo Calabrese¹, Milan Prica¹, Etienne Antoniutti Di Muro¹, and Alberto Montresor²

¹ Dip. Elettrotecnica, Elettronica ed Informatica, Università di Trieste (Italy).

² Dipartimento di Scienze dell'Informazione, Università di Bologna (Italy).

Abstract. Group communication is one of the main paradigms for implementing replication middleware. The high run-time costs of group communication may constitute a major performance bottleneck for modern enterprise applications. In this paper we investigate the applicability of *message packing*, a technique originally proposed by Friedman and Van Renesse in 1997 for improving the performance of group communication, to modern hardware and group communication toolkits. Most importantly, we extend this technique with a policy for varying the packing degree automatically, based on dynamic estimates of the optimal packing degree. The resulting system is *adaptive* in that it allows exploiting message packing efficiently in a dynamic and potentially unknown run-time environment. Several case studies are analyzed.

1 Introduction

Replication of COTS components is now a widespread way for enhancing dependability of services. Group communication is one of the key middleware technologies for supporting replication [7,12,14,6,2,8,3,9,10]. Mainly due to its powerful guarantees, however, group communication usually entails a high cost that plays a decisive role in determining the performance of a replicated application. In our test environment, for example, a non-replicated web service running in Tomcat can sustain a throughput of approximately 300 operations per second. On the other hand, in the same environment, a 3-way replicated application that *only* multicasts and delivers 2000-byte messages with Spread [1] (total order with safe delivery) reaches 100% CPU usage, thereby saturating the system, at a throughput very close to the above. Since a replicated web service requires at least one multicast per operation, it is easy to see that group communication may constitute a major bottleneck for the replicated implementation.

In the attempt of shifting such bottleneck up to higher values, we are investigating techniques for improving the efficiency of existing group communication systems. This paper presents our results in this area. Motivation for this work can be found in the ADAPT project (an EU-funded joined effort between several institutions across Europe and Canada, see <http://adapt.ls.fi.upm.es/adapt.htm> for more details) whose aim is to develop middleware for constructing replicated services based on the

^{*} This work is supported by Microsoft Research (Cambridge, UK) and the EU IST Programme (Project ADAPT IST-2001-37126).

J2EE framework. This goal is part of the broader objective of ADAPT, namely the construction of virtual enterprises out of collections of “basic services” hosted at different, potentially remote, enterprises. In this framework, we are using replication based on group communication as a mean for improving availability and dependability of such basic services. The group communication infrastructure used is Spread, augmented with a thin Java layer on top (as explained later).

The starting point of our work is the proposal by Friedman and Van Renesse [4], in which they demonstrated that *message packing* can significantly improve throughput of total-order protocols. This technique simply consists in buffering application messages for a short period of time before actually sending them as a single message, in order to reduce the overhead caused by the ordering protocol. Their experiments are based on 1997 hardware and, in particular, 10 Mbps Ethernet. Few experiments made it immediately clear that message packing can be very effective even with more modern hardware, including 100 Mbps Ethernet, and even with a group communication system based on a client-daemon architecture (unlike the one used in the cited work).

In this work, however, we exploit message packing in a way quite different from that in [4]. First, we buffer messages until the desired *packing degree* (number of buffered messages) has been reached, irrespective of the amount of time that a message has spent in the buffer. This approach enables us to gain deeper insight into the relationship between *throughput*, *CPU usage*, *latency* and *packing degree*. Of course, a practical implementation will have to introduce an upper bound to the time a message spends in the buffer, otherwise latency could grow excessively and the system could even stop sending messages (the trade-off between throughput and latency is discussed later in the paper). Second, we have defined an *adaptive* policy for changing the packing degree at run-time and automatically. This is a key issue because the packing degree yielding the best performance depends on a number of factors, including characteristics of the message source, message size, processing load associated with each message, hardware and software platform. Not only these factors can be potentially unknown, they can also vary dynamically at run-time. Selecting the packing degree once and for all can hardly be effective. With our policy, the system automatically determines a packing degree close to the value that happens to be optimal in that specific environment (at least for the cases that we have analyzed exhaustively, detailed in the paper). Moreover, the policy has proven to be robust against occasional variations of additional CPU load induced by other applications.

The resulting behavior of the system is as follows: (i) When the source injects a “low” load, message packing remains inactive. (ii) In the case of a “medium-to-high” load, message packing starts to act leading to higher delivered throughput and *decreased CPU usage*. (iii) In the case of a “very high” load, CPU usage reaches 100% anyway but message packing leads to a higher delivered throughput. Our adaptive policy hence helps the system to *automatically* increase the bottleneck point induced by group communication. Although the effectiveness of our proposal will have to be evaluated within a complete replication solution, we believe that these results are encouraging and these features could be very important in the application domain of interest in ADAPT.

The rest of the paper is organized as follows. Section 2 introduces our system and describes the testing methodology used later in the paper. In Section 3, our adaptive

policy for message packing is presented. Section 4 evaluates the effectiveness of the adaptive policy, by describing several tests under various conditions, while Section 5 discusses our results and concludes the paper.

2 The System

2.1 Group Communication System

Our group communication system, called JBora, consists of an harness around Spread. For the purpose of the present discussion, JBora supports an execution model very similar to the Extended Virtual Synchrony model [11] of Spread. The key difference is that JBora supports a notion of primary partition and does not make transitional views visible to applications. From the implementation point of view, JBora consists of a thin Java layer on top of the Java interface to Spread. One JBora multicast maps to exactly one Spread multicast and the size of the two multicasts is the same, except for a 4-byte JBora specific header. This header is inserted without performing an additional memory-to-memory copy beyond those already performed by the Java interface of Spread. When Spread delivers a multicast to JBora, the multicast is immediately deliverable (except when the multicast is delivered in a transitional view, but this condition does not occur in our experiments). One Spread daemon runs on each replica. A JBora application running on a given node connects to the Spread daemon on that node.

Message packing has been implemented by slightly modifying the portion of JBora that implements the operation `mCast(m)`. Rather than invoking the multicast operation of Spread immediately, `mCast(m)` inserts `m` in a *packing buffer*; then the operation may return either immediately or after multicasting the entire packing buffer as a single Spread multicast. The portion of JBora that implements the `Receive()` operation has been modified to unpack received messages as appropriate. Transmission of the packing buffer occurs when the number of messages in it equals the current value for the *packing degree*, denoted as *pack*. This value can be either defined at configuration time and kept constant across the entire execution, or it can be adjusted dynamically based on observed execution statistics (see Section 3). Of course, a practical implementation of message packing has to include further conditions for triggering transmission of the packing buffer. For example, if the number of buffered messages was smaller than *pack* and then the source stopped generating new messages, then the packing buffer would be never transmitted. This topic will be discussed in Section 5.

2.2 Operating Environment

The operating environment consists of a network of Dell Optiplex GX300 (PIII 800MHz, 512 MB RAM), connected to a 100 Mbps switched Ethernet and running Sun Microsystems' JDK 1.4.0 over Windows 2000 Professional. Each node, hereinafter *replica*, is equipped with JBora.

Each replica runs an application written in Java that maintains a simple replicated logging system. The application consists of two threads: the *source* thread generates messages to be multicast through JBora, while the *receiver* thread receives messages

from JBora and writes them into a MySQL database local to the replica. Writes on the database are all done into the same database table. This architecture has been adopted to emulate a simplified, yet realistic replicated three-tier application, where one member multicasts the requests received from clients to all replicas, which execute them by interacting with the database tier. Of course, the performance figures that have been obtained depends on the combination of the various pieces of software present in the system, but we have verified in all experiments below that the bottleneck is indeed the group communication system, not the database.

Each experiment below refers to a system composed of three replicas where only one of them generates messages. We focussed on a small number of replicas because the use of group communication that we are pursuing in ADAPT is for improving fault-tolerance and we believe that, in practical environments, only small replication degrees are likely to be used. We focussed on a single replica that generates messages only for restricting the number of parameters to investigate.

Our experiments are based on sources quite different from those in [4] and much closer to our needs. First, we used total order with *safe* delivery (also called *uniform* delivery). These are the strongest delivery guarantees normally offered by group communication platforms, and also those that are most demanding at run-time. We intend to design replication algorithms based on safe delivery because, without this guarantee, coping with certain failure patterns correctly would require complex and costly actions (e.g., when the sender of a multicast is the only replica that receives that multicast [5]). Second, we considered sources that generate messages continuously or at bursts (alternating a burst with a sleeping time). That is, unlike [4], we do not constrain the generation of new messages by the delivery of messages multicast by other replicas. This is because in our intended application domain generation of new multicasts is triggered by the arrival of operation requests from remote clients, i.e., an event that can occur potentially at any time and usually does not depend on the arrival of multicasts from other replicas. The resulting scenario simulates a situation in which on the sending replica there are always new messages waiting to be multicast. We have implemented a flow control mechanism that suspends the source thread when the load injected into the group communication system is excessive (without this mechanism, the sending replica takes an exception and is forcibly expelled from the group).

2.3 Measurements

For each experiment we have measured *throughput*, *latency* and *CPU usage*. Throughput has been measured as L/N , where L is the time interval between the receiving of the last message and the receiving of the first message. This time interval has been measured at the receiving thread of the sending replica. Quantity N is the number of messages in the experiment run, approximately 25000 in each case. We have not used the standard timer available in Java through the `System` class, because its resolution (approximately 16 msec) was not sufficient for our measurements, in particular for those of latency. Instead, we have used a publicly available timing library that exploits system-specific hooks and allows measuring time intervals with a resolution of 1 microsec [13].

The latency for an experiment run is the average latency amongst all messages of that run. The latency of each message has been measured at the sending replica, as follows.

Table 1. Performance without message packing.

<i>Message size (bytes)</i>	<i>Throughput (msg/sec)</i>	<i>Latency (msec)</i>
100	596	11.07
1000	453	7.08
10000	114	47.28

Table 2. Maximum performance (throughput) obtained with message packing.

<i>Message size (bytes)</i>	<i>Optimal pack</i>	<i>Throughput (msg/sec)</i>	<i>Improvement</i>	<i>Latency (msec)</i>
100	28	5666	9.51	55.24
1000	11	1577	3.48	28.16
10000	3	201	1.76	29.37

The sender thread of the sending replica reads the timer immediately before invoking the Spread multicast operation and inserts the corresponding value in the message to be multicast. The receiver thread of the sending replica reads the timer as soon as it has received a message. The difference between this value and the one contained in the message is the latency value for that message. Note, the time spent in the packing buffer is taken into account for evaluating the latency of each individual message. Average CPU usage has been estimated by visually inspecting the task manager of the Windows 2000 operating system.

3 An Adaptive Policy for Message Packing

Our first suite of experiments used a source thread that *continuously* generates fixed-size messages, putting JBora under stress. The results obtained with message packing disabled are shown in Table 1, for three different message sizes (100,1000,10000). These results constitute the baseline for comparing the results obtained through message packing. Then we made a number of experiments with message packing enabled. In each experiment we kept *pack* constant. The results for 1000-byte messages are in Figure 1. It can be seen that throughput increases substantially, reaching a maximum of 1577 msg/sec with *pack* = 11. This represents an improvement of 3,48 times over the throughput without packing. For sake of brevity, we omit the figures for 100-byte and 10000-byte messages: the curves have the same shape as Figure 1 with numerical values that depend on the message size. A summary is given in Table 2. In all cases throughput increases substantially, at the expense of latency (see also Section 5).

Although message packing may be very effective in improving throughput, a key problem is determining the suitable value for the packing degree *pack*. Our experiments clearly show that the optimum value greatly depends on the message size. Moreover, a realistic source will generate messages of varying sizes. Finally, and most importantly, the effect of message packing may greatly depend on a number of factors that can vary dynamically and usually cannot be predicted in advance, for example, the load on replicas induced by other activities and the specific hardware/software environment. Determining one single value for *pack* once and for all can hardly be effective.

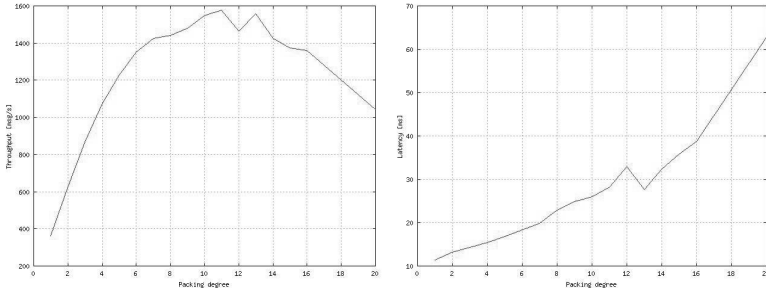


Fig. 1. Throughput (left) and latency (right) with varying packing degrees (msg size is 1000 bytes).

We have implemented a simple and inexpensive *mechanism* for varying *pack* dynamically. Each replica measures the throughput with respect to multicasts generated by that replica at regular intervals, every T_a seconds (the throughput is averaged over this interval). Based on these observed statistics, each replica may vary *pack* dynamically, trying to adapt to the features of the operating environment. In all of the experiments reported here we set $T_a = 5\text{sec}$ and we updated the packing degree based on the last two measures, i.e., every 10 sec. The problem, of course, is determining an effective *policy* for exploiting this mechanism.

We experimented with policies that implement the following basic rules (*throughput_i* denotes the *i*-th throughput measurement, *pack_i* denotes the *i*-th packing degree):

1. Initially, *pack* = 1 (i.e., no packing enabled);
2. *pack* may only be incremented by 1 or decremented by 1;
3. *pack* $\in [1, \text{packmax}]$;
4. $\text{throughput}_i = \text{throughput}_{i-1} \implies \text{pack}_{i+1} := \text{pack}_i$ (steady state);

The issue is defining the *update rule* that varies *pack* so as to improve throughput. From the shape of the curves throughput vs. *pack*, it would appear that defining such rule is simple: one could simply increase *pack* when throughput increases and decrease *pack* otherwise:

- $\text{throughput}_i > \text{throughput}_{i-1} \implies \text{pack}_{i+1} := \text{pack}_i + 1$
- $\text{throughput}_i < \text{throughput}_{i-1} \implies \text{pack}_{i+1} := \text{pack}_i - 1$

The resulting simple policy indeed converges quickly to the value for *pack* that is optimal for the specific message size that characterizes the source. By optimal value we mean the one that we have determined previously, by exhaustive testing (e.g., for 1000-byte messages the optimum value is 11). This is a valuable result because, of course, the system does not know that value but finds it automatically. The quickness in reaching this value depends on how frequently the throughput measurements are taken.

Unfortunately, this policy is not sufficiently robust against occasional throughput variations, induced for example by short additional loads. In many executions, *pack* falls back to its minimum value 1. The reason is as follows. Suppose the optimal value has not been reached yet and throughput_i is lower than throughput_{i-1} because of a

transient phenomenon out of control of the group communication system. In this case, the packing degree would be lowered. At this point it is very likely that the next measurement will show an even lower throughput, thereby ending up quickly with $pack = 1$.

It is also possible that throughput collapses because $pack$ oscillates around excessively high values. To realize this, consider Figure 2. Suppose the system be characterized by curve B and the packing degree has reached its optimal value pB . Next suppose that, due to some additional load, the system be characterized by curve A. The next measure will show that throughput has decreased, thus $pack$ will be decremented to $pB - 1$. The next measure will then show that throughput has increased, thus $pack$ will be incremented again at pB . Since this increment will cause throughput to decrease, at this point the value of $pack$ will keep on oscillating around pB , a value that may be largely suboptimal in curve A. Note, phenomena similar to those just discussed could occur even if the source changed the message size during the run.

For these reasons, we experimented with a simple refinement of the above update rule. The basic idea is this: one has to make sure that when $pack$ starts to decrease, it may continue decreasing only if throughput grows — i.e., only when $pack$ is indeed greater than the optimal value corresponding to the peak throughput. Otherwise, $pack$ should no longer decrease and should increase instead. We implement this idea with the following update rule:

- $pack_i \geq pack_{i-1} \implies // \text{Increase pack when throughput increases}$
 - $throughput_i > throughput_{i-1} \implies pack_{i+1} := pack_i + 1;$
 - $throughput_i < throughput_{i-1} \implies pack_{i+1} := pack_i - 1;$
- $pack_i < pack_{i-1} \implies // \text{Decrease pack when throughput increases}$
 - $throughput_i > throughput_{i-1} \implies pack_{i+1} := pack_i - 1;$
 - $throughput_i < throughput_{i-1} \implies pack_{i+1} := pack_i + 1;$

It is simple to realize that, as confirmed by our experiments, this policy prevents the instability behaviors described above. Short transient loads may provoke a decrease of the packing degree, but not its collapsing to the minimum value. Once the additional load has disappeared, the packing degree converges again to its previous value. Similarly, the packing degree does not oscillate around excessively high values. The policy is thus quite robust. All the results presented later are based on this policy.

4 Evaluation of the Adaptive Policy

4.1 Continuous Source

The most demanding test for a group communication system is given by a source thread that *continuously* generates new multicasts. We have evaluated this scenario with both messages of fixed size, and with messages of variable size. All the results in this section corresponds to a CPU usage close to 100%. That is, nearly all of the CPU time on the sending replica is spent for propagating messages (recall that each message is logged on a MySQL database, though).

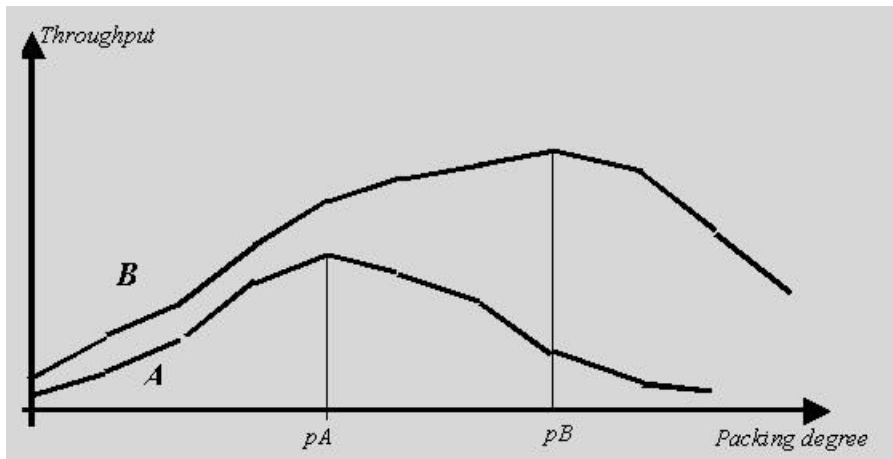


Fig. 2. Throughput vs. Packing degree for different loads: curve A corresponds to an higher load than curve B (curves for differing message sizes and/or additional load have the same shape).

Messages with Fixed Size. With the policy enabled and 1000-byte messages, the packing degree oscillates between 10 and 12 and the average throughput is 1338 msg/sec. This corresponds to 85% of the throughput obtained with the packing degree statically set to its optimal value 11. It also corresponds to almost a 300% throughput improvement over the system without packing enabled. Latency is 19.58 ms.

The time it takes to *pack* for reaching the 10-12 range from its initial value 1 is approximately 100 seconds. The reason is, we update the packing degree every 10 seconds and we can only change it by 1 at every step. We did not experiment with more aggressive policies attempting to shorten this interval. This could be done with shorter update intervals, e.g., 2-3 seconds. We believe that altering the packing degree by more than one unit could make the policy less stable with more realistic sources and environments. We leave this topic open for further investigation and will not mention this issue any further here.

The reason why the packing degree *pack* does not remain constant but oscillates around the optimal value is because consecutive throughput measures, in practice, will never show exactly the same result. We could filter this effect out by updating *pack* only when the difference between consecutive measures falls outside some threshold. Although this approach could increase the average throughput further, it would also introduce another parameter to define and to possibly tune. We preferred to avoid this in the attempt to make a system that requires no magic constants and can tune itself automatically, albeit in a slightly sub-optimal way.

Messages with Variable Size. We have performed experiments with a source that injects messages continuously, but with differing sizes. Below we present results for the case in which the source generates 300.000 1000-byte messages followed by 300.000 3000-byte messages and then repeats this pattern indefinitely.

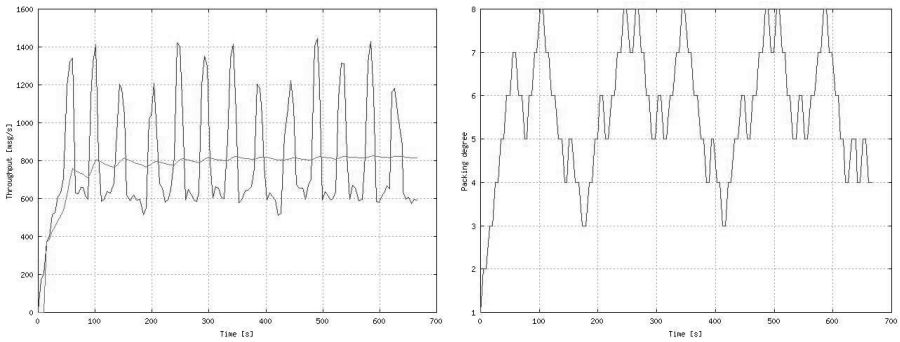


Fig. 3. Average throughput and “instantaneous” throughput over time (left). Packing degree over time (right).

First we have performed a set of experiments for measuring the throughput as a function of *pack*, by keeping the packing degree constant in each run. We have found that throughput without packing is 269 msg/sec, that the maximum throughput is obtained with *pack* = 6 and corresponds to 901 msg/sec. Then we have exercised the system with our policy enabled. Figure 3-left shows the throughput measurements. The flat line shows the average throughput, averaged since the beginning of the experiment. The other line shows the “instantaneous” throughput, i.e., averaged over the last 5 seconds. It can be seen that the average throughput reaches a value close to 800 msg/sec in less than 1 minute and then remains stable despite the variations of the source at around 810 msg/sec. This value corresponds to approximately 90% of the maximum throughput, obtained with *pack* immutable and fixed a priori to 6. It also corresponds to an almost 300% throughput improvement over the system without packing. Figure 3-right shows the variations of *pack* over a short time interval.

4.2 Bursty Sources

We have performed experiments with a *bursty* source. The source thread generates a burst of 15 1000-byte messages, sleeps for 20 msecs and then repeats this pattern indefinitely. These experiments are important not only because the source is less extreme than the continuous source discussed above, but also because in this case the CPU usage is smaller than 100%. This scenario should be closer to practical applications where substantial resources are required beyond those consumed by group communication, for example, replication of J2EE components.

These experiments shows an important finding: CPU usage varies with the packing degree *pack* in a way that is roughly opposite to throughput. That is, the packing degree resulting in peak throughput also results in minimum CPU usage. It follows that message packing may greatly help in improving the overall performance of a complete application, because it contributes to decrease the CPU time required by the replication infrastructure. Another important finding is that our policy for adapting the packing degree automatically works also in this case and indeed decreases significantly the CPU usage.

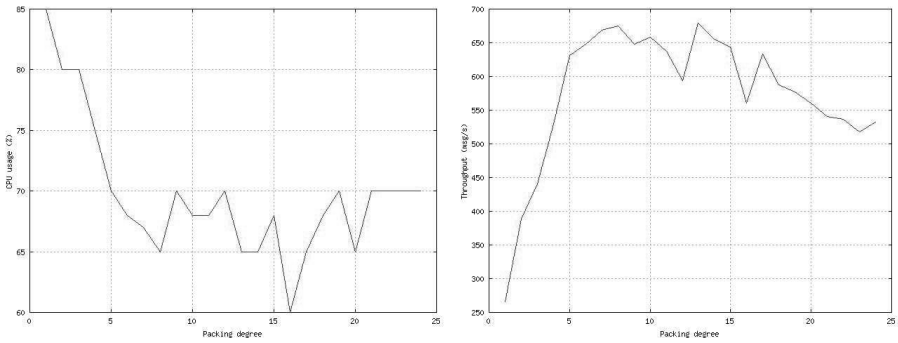


Fig. 4. Average CPU usage (left) and average throughput (right) over packing degree. Bursty source: 15 messages and then 20 msec sleeping time.

First we have performed a set of experiment runs by keeping the packing degree constant in each run. The results are in Figure 4. Without packing enabled, CPU usage is 85% and throughput is approximately 260 msg/s. With packing enabled, the maximum average throughput is obtained with $pack = 8$ and corresponds to 660 msg/sec. Note that in this situation CPU usage has dropped to 65%.

Then we have run the system with our automatic policy enabled. The average throughput reaches a value close to 430 msg/sec in slightly more than 1 minute. The CPU usage remains below 60%. The packing degree remains stable around two values: 6 for some time intervals and 10 for some others. This behavior is probably due to the fact that the curve throughput vs. packing (Figure 4-right) does not exhibit a single peak and is more irregular than the curves analyzed in the previous section. In summary, the policy increases the throughput by 165% and lets the CPU usage drop from 85% to less than 60%.

4.3 Short Bursts

Finally, we have investigated the behavior of the system with very short bursts of 1000-byte messages. We made a number of experiments varying the number of messages in each burst and the sleeping time between bursts. Roughly, we have found that as long as the rate of generation of new messages is above 250 msg/s, our automatic policy still increases throughput and decreases CPU usage. Below such rate our policy has no effect. Two of the combinations burst length/sleeping time where the policy has effect are given in Table 3.

It is not surprising that when the throughput injected into the system is sufficiently low, message packing has effect neither on throughput nor on CPU usage — it can only increase latency. However, the overall result is significant: when the injected load is sufficiently low, the system is capable of sustaining such load autonomously; when the injected load is not so low, our adaptive policy automatically helps the system in sustaining that load, by shifting the group communication bottleneck to higher loads.

Indeed, these experiments allowed us to identify an issue where our policy needs some refinement. The curve throughput vs. packing degree shows a step when $pack$

Table 3. Results with bursty source, very short bursts.

Source	Scenario	Throughput (msg/sec)	Latency (msec)	CPU usage
5 msgs every 20 msec	No packing	154	10	60%
	Policy enabled	236	18	45%
2 msgs every 5 msec	No packing	285	5.9	65%
	Policy enabled	319	11	45%

becomes greater than 1 and then remains more or less flat for a wide range of values of *pack*. It follows that, with the current policy, the packing degree exhibits fairly wide oscillations. Although the resulting behavior is still satisfactory, it seems that a smarter policy is required.

5 Concluding Remarks

Friedman and Van Renesse demonstrated in 1997 that message packing can be very effective in improving throughput of group communication systems. Our experiments show that this argument still holds with more modern hardware (including 100 Mbps Ethernet) and when safe delivery is required. Most importantly, we have shown that one can exploit message packing *adaptively*, by means of a simple policy that dynamically matches the packing degree to the specific and potentially unknown characteristics of the message source. Our proposed policy is based on a simple and inexpensive mechanism and has proven to be robust against dynamic and unpredictable changes in the run-time environment.

Of course, message packing is most effective when the source is demanding. In this respect, the best results are obtained when the source injects a very high load for a very long time. However, we have seen that message packing is effective even with sources that inject relatively short message bursts.

We have investigated the effects of message packing even in scenarios when the CPU usage is well below 100%, to simulate a situation in which the group communication system is part of a complex and demanding application based on replication, e.g., replication of J2EE components. We have observed that even in this case message packing can improve throughput substantially and, most importantly, while *decreasing* CPU usage. The main drawback of message packing is that it tends to increase latency, presenting an important trade-off between this quantity and throughput. While our proposed mechanism and policy for message packing are certainly to be evaluated in the context of a complete replication solution, we believe they constitute indeed a promising approach.

To put this claim in perspective, we report data collected from a prototype of a replicated web service that we have developed on top of JBora. The service implements a counter private of each client. Updates to the counter are multicast to each replica. The service implementation does *not* support message packing yet. Clients access the service through SOAP over HTTP. We simulated a varying number of clients with a publicly-available tool (<http://grinder.sourceforge.net/>). Each client executes an endless loop in which it sends a request and parses the matching response. The data

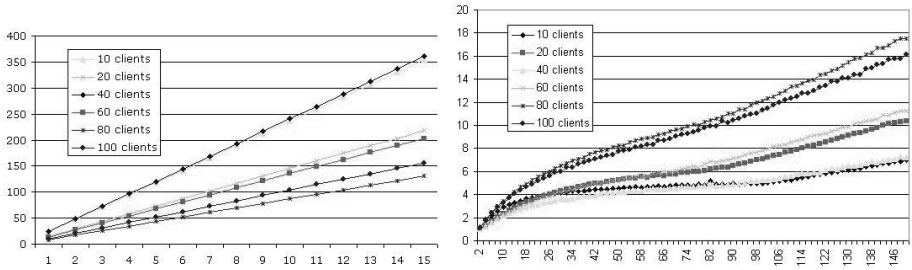


Fig. 5. Average time for filling a specified packing degree (left) and average number of multicasts as a function of the packing interval (right). Times are in ms.

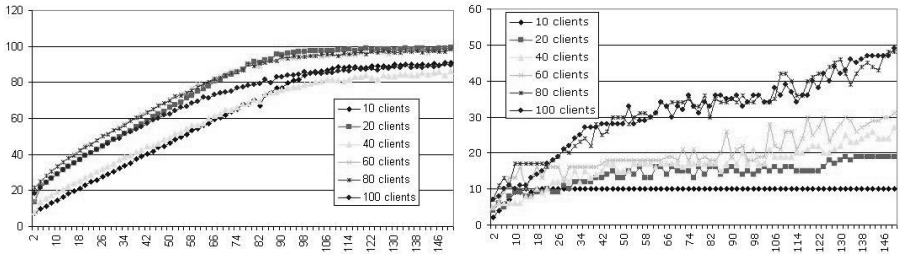


Fig. 6. Percentage of non-empty packing intervals (left) and maximum number of multicasts (right) as a function of the packing interval. Times are in ms.

below refer to the same environment used previously, 3 replicas, clients and replicas are on the same Ethernet and all operations are updates. In the range 80-120 clients, throughput grows linearly from 80 to 225 operations/s whereas latency decreases from 900 ms to 416 ms. In the range up 200 clients, throughput remains in the range 200-250 operations/s whereas latency remains in the range 400-550 ms.

We instrumented this replicated web service so as to record the time instants at which each multicast operation is invoked. Then we analyzed off-line these data to obtain a rough indication of whether message packing can realistically be exploited or not. Figure 5-left shows the average time it would take to collect a number of multicasts equal to the (hypothetical) packing degree, as a function of the packing degree. For example, with 100 clients one could collect 8 messages in 83 msec, on the average. By comparing these data with the previous results and having observed that in this case the message size is 810 byte (slightly smaller than 1000 bytes, the size used in the previous experiments), we note that insisting on achieving the optimal packing degree could not be realistic, because the time spent in the packing buffer could grow excessively — well beyond 100 ms. On the other hand, it also appears that message packing can indeed be applied significantly.

JBora is being extended in order to trigger transmission when either the current packing degree *pack* has been reached or one of the messages has been in the packing buffer for a time larger than a predefined *packing interval* $pack_{TIME}$. The duration of the packing interval is defined statically, depending on the latency requirements of the

application (see also below). This additional condition is capable to handle very irregular sources without introducing unacceptably high delays within the packing buffer (as well as sources that could not even fill the packing buffer). To gain insights into this issue, we analyzed the above data as if the packing interval was the only condition that triggers the transmission. The results are in Figure 5-right, that shows the average number of multicasts in each (hypothetical) packing interval. The average has been done with respect to the packing intervals that contain at least one multicast. For example, with $pack_{TIME}$ set to 70 ms and 100 clients, each non-empty packing interval would contain 9 multicasts, on the average. Figure 6-right shows the maximum number of multicasts rather than the average, whereas Figure 6-left shows the percentage of non-empty packing intervals. These data indeed confirm that message packing can be realistically exploited, albeit probably in a sub-optimal way.

As future work, we intend to complete the implementation of message packing in order to fully evaluate its effectiveness in the above replicated web service. We are also going to investigate whether allowing the length of the packing interval to vary dynamically and automatically between two or three predefined values is both simple and desirable. A further issue that deserves more investigation is whether a more aggressive policy that follows more quickly dynamic variations in the run-time environment is really required and can be implemented without detailed, a-priori knowledge about the environment itself.

References

1. Y. Amir and J. Stanton. The spread wide-area group communication system. Technical Report CNDS 98-4, Johns Hopkins University, 1998.
<http://www.spread.org>.
2. Y. Amir and C. Tutu. From total order to database replication. In *Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2002.
3. R. Friedman and E. Hadad. A group object adaptor-based approach to CORBA fault-tolerance. *IEEE Distributed Systems Online*, 2(7), November 2001.
4. R. Friedman and R. van Renesse. Packing messages as a tool for boosting the performance of total ordering protocols. In *Proc. of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC '97)*, 1997.
5. C. Karamanolis and J. Magee. Client-access protocols for replicated services. *IEEE Transactions on Software Engineering*, 25(1), January/February 1999.
6. B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems*, 25(3): 333–379, 2000.
7. P. M. Melliar-Smith L. E. Moser and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2): 81–92, 1998.
8. S. Labourey and B. Burke. JBoss clustering. Technical report, The JBoss Groups, 2002.
9. S. Mishra, L. Fei, X. Lin, and G. Xing. On group communication support in CORBA. *IEEE Transactions on Parallel and Distributed Systems*, 12(2), February 2001.
10. G. Morgan, S. Shrivastava, P. Ezhilchelvan, and M. Little. Design and implementation of a CORBA fault-tolerant object group service. In *Proceedings of the 2nd IFIP International Working Conference on Distributed Applications and Interoperable Systems*, 1999.
11. L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 56–65, 1994.

12. Y. Ren, D. Bakken, T. Courtney, M. Cukier, D. Karr, P. Ruble, C. Sabnis, W. Sanders, R. Schantz, and M. Seri. AQuA: An adaptive architecture that provides dependable distributed objects. *IEEE Transactions on Computers*, 52(1): 31–49, January 2003.
13. V. Roubtsov. My kingdom for a good timer! *Javaworld*, January 2003.
<http://www.javaworld.com>.
14. S. Bagchi Z. Kalbarczyk, I. Ravishankar and K. Whisnant. Chameleon: A software infrastructure for adaptive fault-tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10(6): 560–579, June 1999.

DADO: A Novel Programming Model for Distributed, Heterogenous, Late-Bound QoS Implementations

Eric A. Wohlstadter and Premkumar T. Devanbu

Software Systems Research Laboratory
Department of Computer Science
University of California at Davis
Davis, CA 95616, U.S.A.
{devanbu, wohlstad}@cs.ucdavis.edu

Abstract. Quality of service implementations, such as security and reliability, are notoriously difficult for software engineers. They have fragmented, cross-cutting implementations, with elements required in application components on both client and server sides. Heterogeneity and distribution make things even more difficult. To cap it all, precise requirements, specially for security, are often deployment-specific. DADO is a new programming model, with roots in aspect-oriented programming, that aims to improve matters. The goal of DADO is to provide a CORBA-like programming model to the developers of distributed, heterogeneous QoS features. DADO comprises a modeling language, a deployment language, code generators, and a run-time environment. DADO allows QoS features to be modeled at the IDL level. Communication between QoS elements is explicitly modeled, as is the interaction between QoS elements and applications (*viz.*, advice). Model-driven code-generation handles the mundane details of marshaling, despatch, and heterogeneity. DADO has a separate deployment language that allows QoS services to be deployed against applications. The DADO run-time also has several interesting features: clients can dynamically respond to QoS features at the server by deploying matching client-side service elements at run-time. DADO supports various interception mechanisms, such as proxies, interceptors, and code transformation, and all mechanisms are inter-operable. DADO currently works on TAO and JacORB, and supports QoS implementations in both Java and C++.

1 Introduction

Consider a distributed medical application, wherein clients obtain prescriptions from doctors. The attendant security policy requires that a client must obtain a credential from a credential server (as in Kerberos; this eliminates the need for the client to reveal his password to anyone but the credential server, and also does not require the doctors/clients to reveal the security policy to the authentication server) and present this credential to authenticate themselves to

a doctor. To use the doctor, the clients must first register themselves with a doctor for a period of time (this is to prevent clients from seeking duplicate prescriptions from multiple doctors). While registered with a doctor, a client may not use other doctors. The client can then obtain prescriptions.

Such a policy presents serious challenges for developers using middleware such as CORBA. The policy requires modifications to both client-side (to obtain credentials) and server-side (to validate credentials/registration, and also register clients), and requires an extra bit of data (the credential) to be sent across. Existing options for implementing such a service are unattractive. Modifying the interface, and weaving the policy code into the application tightly binds the security policy into the application and tangles application code, vitiates separation of concerns, precludes the reuse of the policy code elsewhere, and complicates (likely) future modifications to the security policy. Reflective implementations ([1,2,9]) do allow separation of concerns, and a componentizable, reusable implementation; however, the use of reflection reduces the utility of static type-checking, and usually requires hand-written marshaling, demarshaling and dispatching code.

DADO provides a different approach to programming security and other QoS features: the design philosophy is *to bring to service developers the same convenient programming model that CORBA brings to developers of distributed heterogeneous software*.

2 DADO

The Dado programming model comprises languages, tools, and run-time enhancements.

Languages. There are two languages: the DADO interface description language, or *DAIDL*, is used to describe the interface between the service elements and the application components. The interfaces of client-side and server-side service elements are modeled separately. DAIDL models QoS methods, called *advice*, that interact with application methods. The communications between client- and server-side is also explicitly modeled, as *requests*. The second language, DADO deployment language, or *DDL*, is based on Aspectj [7] pointcuts; this language specifies precisely how a QoS feature interacts with an underlying application. For example, we can specify which methods are intercepted by the security QoS advice, and (if need be) which arguments of these methods are to be captured and sent to the security policy implementation.

Tools. Dado has two sets of tools, for *development time* and *deployment time*. At development time, code generators produce stubs and skeletons from the DAIDL descriptions. These provide multi-lingual type environments (currently, for C++ and Java) that allow service developers to write client-side and server-side QoS code in a type safe manner. Marshalling and demarshalling code for QoS data exchange are also generated. The deployment time tools match the QoS DAIDLs against the application IDLs, and based

on the DDL specifications, generate despatch tables that control the invocation of DAIDL advice.

Run-Time Libraries. DADO run-time libraries handle despatch of advice, and of requests. The run-time also handles the dynamic matching (on the client-side) of client QoS features to QoS features deployed at the server.

Figure 1 (reproduced from [15]) describes the typical software process workflow when using DADO to develop and deploy QoS features. The caption on the figure explains the process. We emphasize here that DADO aspires to the CORBA “hourglass” philosophy: a simple, flexible, powerful programming model that can be implemented in a variety of different ways. In particular, DADO allows QoS features to be implemented in a variety of programming languages (currently, we support Java and C++), and allows QoS elements to be inserted (instrumented) into the application code in a variety of different ways. Certainly, in a heterogeneous environment, different implementation and instrumentation technologies can co-exist.

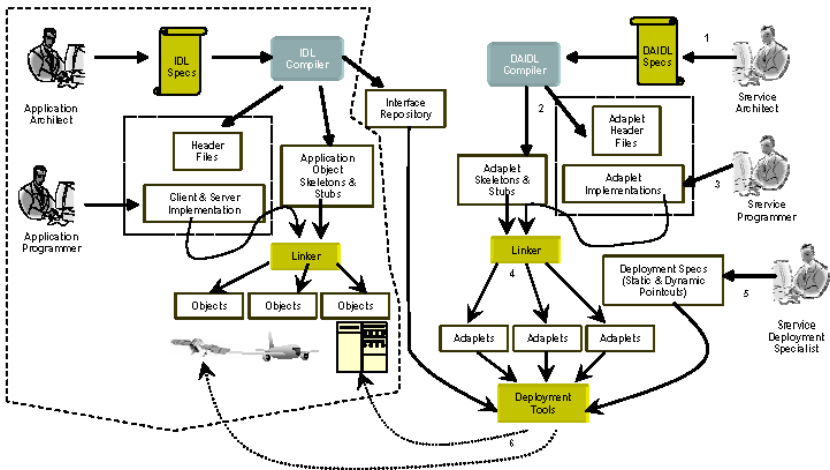


Fig. 1. *DADO Development Process.* The left hand side (within the dotted lines) indicates the conventional CORBA process. On the right, the DADO service development begins (1) with modeling the interfaces to DADO adaptlets using DAIDL; from this the DAIDL compiler generates (2) plumbing code, and typing contexts for adaplet implementations. The programmer writes (3) the adaplet implementations and links to get (4) the adaptlets. Now, the development specialist produces (5) deployment specs, and these are used by deployment tools to install (6) the adaptlets at the proper application object locations. Deployment can occur at compile time, link time, or run-time, depending on the instrumentation technology used (only run-time insertion is illustrated in the figure).

3 Security Example

We now turn to the security policy example presented at the beginning of this paper. The Dado interface description of this QoS feature is shown below. We note that this description simply provides the interface; as in the case of CORBA (and as described above in figure 1) this interface is processed by tools to provide stubs and skeletons; the skeletons then have to be implemented by service programmers.

```
adaplet Wall {
  server {
    request authenticInfo(in Authentic::Token tok);
    request register(in Authentic::Token tok,
                    in long duration);
    advice void check()
              raises(NotRegistered,NoAccess);
  };
  client {
    advice contactAuthentic();
  };
};
```

A QoS feature in DADO (specifically, DAIDL) is termed an *adaplet*. An adaplet has two interfaces, one for the client side, and one for the server side. In this case, the client-side adaplet interface comprises one *advice* method, which intercepts the client-side of application methods that require security policy enactment. The implementation of the advice `contactAuthentic` in this case will get the authentication token, and communicate it to the server side. The server-side adaplet will have the logic to check this token, and also register the client with a particular server implementation. These two operations (receiving and checking the token, and registering a client) are modeled in the server side adaplet interface as *requests*: `authenticInfo` does the former, and `register` the latter. Finally, the advice operation `check` will intercede on application methods requiring security policy enactment.

4 Performance (Cacheing) Example

We present a cacheing example, whereby a client requests string-indexed values (*e.g.* stock quotes) from a server. These values are assumed to periodically change, as the server receives updates from a notification (“push”) service. Here, a client-side adaplet maintains a string-indexed cache of values. If a request can be answered from the cache, the server is never contacted. To complicate matters slightly, we also assume that the client side adaplet can request the server side for a time-to-live (TTL) value. The server side adaplet maintains TTL values for various string indices by intercepting updates and observing update frequency for various string indices (*e.g.*, the “NYSE:HAL” ticker symbol might be updated more often when an oil-producing country is invaded).

```

adaplet Cache
{
    client( A ) {
        request void set_TTL(in long ttl);
        around A get(in string hashCode);
    };
    server      {
        request void ask_for_TTL();
        before void update_TTL();
    };
};

```

We start with an abstract interface to the client and server side adaplet components necessary to carry out the caching service. The client-side includes a request operation to receive TTL values that are “piggybacked”¹, on server responses. Additionally, an around advice is specified to short circuit client requests for which cached data is available. The cache is keyed through the string index. This index is obtained from the application that the cache is deployed on. The type of data stored by the cache can be polymorphic and is bound though the type parameter *A*.

On the server side there is also one request and one advice operation. The request `ask_for_TTL` allows the client side adaplet to signal the server side that it is planning to cache the data in the current invocation response. The before advice `update_TTL` should be triggered whenever application events occur on the server side that invalidate cache data. This allows the server to heuristically adjust its forecasted TTL values. The DDL specification to bind this service to a specific application is shown below. Notice the instantiation of type parameter *A* with `float`.

```

adaplet ClientSideStockCache : Cache
{
    client: Cache < float > {
        pointcut cachedOperation(in string x) :
            call(float StockServer::getQuote(x));
        around cachedOperation(x) :
            get(in string x);
    };
};

```

Now, the abstract interface must be specialized for deployment on both client and server. Specialization is achieved through object-oriented inheritance. In order to cache the result of the `StockServer::getQuote` operation the client side cache type parameter *A* is instantiated as `float`. The pointcut `cachedOperation` matches the invocation to `StockServer::getQuote` and captures the argument to key the cache. Finally this pointcut is applied to the around advice that we want to intercept these invocations.

¹ The DADO runtime packages QoS related messages from client to server (or vice versa) in the *service context* field in the IIOP RPC message


```

adaplet ServerSideStockCache : Cache
{
    server{
        pointcut timedOperation() :
            call(void StockServer::setQuotes(in QuoteList));
        before timedOperation(): update_TTL();
    };
};

```

The server side deployment consists of dispatching the `update_TTL` advice for operations where TTL calculation can be updated. The pointcut `timedOperation` matches the operation `StockServer::setQuotes` and is applied so `update_TTL` will be dispatched before those invocations.

5 Current Status

Currently, DADO works on both Linux and Windows. On both platforms, we support both TAO and JacORB. DADO tooling can generate both C++ (Gnu and Microsoft) and Java. Performance measurements indicate that the overhead of our marshalling and dispatching machinery is reasonably modest. Details are available in [15]. Several sample services have been implemented; including the above two, we have also implemented a generic failover service. We have also experimented with generic adaptations to prevent denial-of-service attacks, and generic implementations of P2P architectural styles. Work in progress includes development of a DADO-like approach for web services, as well as dynamic, co-ordinated deployment of QoS services in complex, distributed, feature-rich and feature-interacting settings.

6 Related Work

There is related work on handling cross-cutting adaptations in heterogenous environments. We have already compared DADO with reflective ORBS [14,8,3]. Interceptors [14,9], and filters [11] can intercept every method, but require marshalling and dispatch to be hand-constructed without the benefit of static typechecking. Proxies and wrappers [5,12] are specific to applications. Containers [10,13] generalize by exploiting code generation, but require new code generators for each new QoS approach. In addition, client-side QoS adaptations are not supported.

Duclos, Estublier, and Marat [4] describe the component virtual machine, (CVM). CVM provides a meta-object protocol for components that allows component adaptation, specific as AspectJ-style pointcuts. DADO is similar in spirit, but complements CVM by allowing component adaptations on both client and server sides. CQoS [6] allows the construction of generic QoS components using generic QoS components, and application/platform dependent interceptors. Cactus provides consistent coordination adaptation of distributed systems using

micro-protocols. DADO exploits on aspect-oriented model for adapting applications with QoS features. CQoS allows per-object QoS bindings, whereas the current implementation of DADO only allows per-POA bindings.

7 Conclusion

DADO provides a convenient programming model for building QoS features such as security, fault-tolerance and billing/usage tracking in a distributed, heterogeneous, setting. DADO can conveniently handle split-context QoS features, and supports piggy-backed messaging from client to server side and vice versa. DADO supports a variety of different instrumentation techniques interoperably. DADO comprises a modeling language, code generation tools, run-time enhancements, and a deployment language. DADO is currently available for both Java and C++ on JacORB and TAO, and works on both Linux and Windows. An unsupported version is available at rickshaw.cs.ucdavis.edu

References

1. G. Blair and R. Campbell, editors. *Reflective Middleware*, 2000.
2. L. Capra, W. Emmerich, and C. Mascolo. Reflective middleware solutions for context-aware applications. *Lecture Notes in Computer Science*, 2192: 126–??, 2001.
3. N. Coskun and R. Sessions. Class objects in som. *IBM Personal Systems Developer*, Summer 1992.
4. F. Duclos, J. Estublier, and P. Morat. Describing and using non functional aspects in component based applications. In *International Conference on Aspect-Oriented Software Development*, 2002.
5. T. Fraser, L. Badger, and M. Feldman. Hardening COTS software with generic software wrappers. In *IEEE Symposium on Security and Privacy*, pages 2–16, 1999.
6. J. He, M. A. Hiltunen, M. Rajagopalan, and R. D. Schlichting. Providing qos customization in distributed object systems. In *Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms*. Springer-Verlag Heidelberg, January 2001.
7. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072: 327–355, 2001.
8. F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, and R. H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*
9. P. Narasimhan, L. Moser, and P. Mellior-Smith. Using interceptors to enhance CORBA. *IEEE Computer*, July 1999.
10. E. Roman, S. Ambler, and T. Jewell. *Mastering Enterprise JavaBeans*. Wiley, 2001.
11. J. Siegel. *CORBA 3 Fundamentals and Programming*. Wiley Press, 2000.
12. T. S. Souder and S. Mancoridis. A tool for securely integrating legacy systems into a distributed environment. In *Working Conference on Reverse Engineering*, pages 47–55, 1999.

13. A. Troelsen. *C# and the .NET Platform*. Apress, 2001.
14. N. Wang, K. Parameswaran, and D. Schmidt. The design and performance of meta-programming mechanisms for object request broker middleware, 2000.
15. E. Wohlstadter, S. Jackson, and P. Devanbu. Dado: Enhancing middleware to support crosscutting services. In *Proceedings of the International Conference on Software Engineering*, Portland, USA, 2003. IEEE.

Many Aspects of Reliabilities in a Distributed Mobile Messaging Middleware over JMS

Eiko Yoneki

University of Cambridge Computer Laboratory
William Gates Building, J J Thomson Avenue
Cambridge CB3 0FD, UK
{eiko.yoneki}@cl.cam.ac.uk

Abstract. Pronto[19] is a middleware system for mobile applications with JMS messaging as a basis in both centralized and decentralized forms. Asynchronous messaging is essential to support reliable communication with mobile devices. An intelligent gateway and smart caching in Pronto support the construction of distributed messaging systems over JMS servers. The main aim of Pronto is to support a reliable and efficient message flow from data source to consumers by applying and comparing different techniques. Pronto provides a solution for mobile application specific problems such as resource constraints, network characteristics, and data optimization, which achieves high reliability in end-to-end communication.

1 Introduction and Background

Computing devices are becoming increasingly mobile. Mobile computing need to deal with more dynamic environments and more resource constraints than traditional desktop computing. In a mobile/wireless network environment, devices have a small ROM/RAM footprint, latency is high, bandwidth is low, connections are frequently interrupted, location of devices changes at any time, and many devices are not programmable. This diversity of clients creates complex environments in distributed systems. It is challenging to provide reliability under such circumstances, especially as the information delivered to mobile devices may be mission critical messages. Thus, middleware communication service is important for integrating hybrid environments with high reliability.

Several communication mechanisms such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) have been used for sharing the workload in distributed systems. Intercommunication is commonly achieved using directed links between tightly coupled senders and receivers, where the message destination must be known at the time of sending; this is difficult with changing destinations or varying numbers of recipients. By contrast, Message Oriented Middleware (MOM) encourages loose coupling between message senders and receivers with a high degree of anonymity, the advantage of taking away static dependencies in a distributed environment. MOM's characteristics (intuitive programming model, latency hiding, guaranteed delivery, store-and-forward) are highly appealing for mobile applications to provide reliable message delivery.

Java Message Service (JMS)[15] is a recent Java technology, providing an API for inter-client communication among distributed applications. The communication mechanism of JMS differs from others such as EJB (Enterprise Java Beans), and CORBA (Common Object Request Broker Architecture), which define a higher level of logic for applications. JMS is a service-oriented API specification, providing an architecture for MOM and prescribing messaging functionality in terms of interfaces and associated semantics. It offers publish-subscribe and point-to-point paradigms and expands previous messaging capabilities. JMS works well in an environment where network connections can break, and where the available bandwidth can vary within a short time. Most JMS products implement a centralized server model. To provide rich JMS functionality, especially persistent message delivery, servers require databases (for storing messages), yet none of the commercial products has successfully implemented JMS in a decentralized model. There have been efforts to construct messaging systems over peer-to-peer networks in distributed format, but none provide enterprise level messaging functionality thus far. Given the characteristics of mobile devices and wireless networks, more work is required for high performance and reliability. Some important design issues are specified below:

- Wireless networks become increasingly packet-oriented. With a packet-oriented bearer such as GPRS (General Packet Radio Service) or UMTS (Universal Mobile Telecommunications System), users typically pay only for the data they communicate. Reducing data size for transmission is crucial.
- Because of low bandwidth, high latency, and frequent disconnections, a middleware should provide an interface to applications that allows for communication to be maintained during disconnected operation. Dependable caching is essential.
- A data source can be interpreted in different formats and semantics depending on the specifications of mobile devices and wireless networks. Semantic transcoding technology [11] should allow for more efficient data flow.
- There are various bearers such as 2G, 2.5G, 3G, Bluetooth, and IEEE 802.11 and many devices are non-programmable. A middleware needs to offer an interface that provides a communication abstraction.

This paper presents **Pronto**, a middleware system for mobile applications, that is briefly introduced in [19]. The basis of Pronto is a MOM based on JMS in both centralized and decentralized forms. The decentralized form is called **Serverless JMS**. Pronto also introduces an intelligent **Gateway** with store-and-forward messaging [18] for reliable and efficient transmission between mobile applications and servers, taking advantage of plug-in components for caching, device-specific transport, compression, semantic transcoding, and message transformation, as well as supporting disconnected operation. **SmartCaching** component provides generic caching in an N-tier architecture, an essential function of Gateway. Caching reduces the communication between mobile devices and data source, leading to gain high reliability. Constructing a distributed system by deploying multiple gateways over JMS servers controls the

flow of messages, which limits unnecessary incoming network traffic to mobile environments. The original data source should be made abstract to provide compaction, then data should be evaluated whenever necessary. By going through gateways, messages need to be more localized. This gives optimized reliable and efficient message flow.

1.1 Delivery Semantics and Reliability

In messaging middleware systems, in particular when considering communication models and protocols, precise specification of the semantics of a delivery is an important issue. Delivery guarantees are often limited by the behavior of lower communication layers, down to the properties of the network itself, limiting the choice of feasible semantics. Reliable point-to-point communication is usually achieved by applying TCP. In messaging middleware, messages may be sent without generating a reply, and they may be processed hours after having been sent. The communicating parties do not handle how messages are transmitted and when they are processed. Thus, the messaging system must provide guarantees both in terms of reliability and durability of the messages. It is not sufficient to know that a message has reached the messaging system that sits between the producers and consumers. It is necessary to guarantee that the messages will not be lost due to failure of that messaging system. Messaging systems need to provide various guarantees regarding reliability and fault tolerance ranging from best-effort to guaranteed and timely.

Persistence is generally presented in messaging systems in a centralized model and stores messages until consumers process them. It is based on store-and-forward messaging, i.e. the provision of reliability over a network where connections may be dropped and restored. It is a trade-off because more processing time is required for store-and-forward. Therefore a compromise to accomplish the balance is important. On the other hand, distributed messaging middleware in a decentralized model generally do not offer persistence since messages are typically replicated upon dissemination. By sending one copy to each subscriber, the system provides some degree of fault tolerance. However, a subscriber that has unstable communication condition may not be able to obtain missed messages when recovering. In distributed messaging systems such as Scribe/Tapestry can repair the multicast tree by periodically sending heartbeat messages. Other systems use a mesh network such as Buyeux. The redundancy of a mesh network can guarantee reliable delivery when a path is broken due to a faulty broker. Gryphon achieve reliability in the similar manner. While sometimes a high throughput is important and a low degree of reliability is acceptable, some applications set higher priority over throughput. Messaging systems therefore should provide different qualities of services to meet the demands of different applications. The delivery semantics for notifications offered by existing systems can be roughly divided into two groups, see also Section 3 for JMS Message Flow:

- **Unreliable delivery:** Protocols for unreliable delivery give few guarantees. These semantics are often used for applications where the throughput is the most important issue, but the loss of certain messages is not fatal for the application.
- **Reliable delivery:** Reliable delivery indicates that a message will be delivered to every subscriber despite the failure.

1.2 Many Aspects of Reliability

As distributed applications grow both in size and complexity, there is an increasing demand to improve reliability. Different clients of messaging middleware will have different degree of requirements regarding to reliability and quality of service (QoS) guarantees provided by the middleware. Fault-tolerance mechanisms have to be considered within the messaging middleware design, thus isolated network or component failures do not affect the entire systems. Techniques such as persistent messages and replication help to achieve a more robust messaging middleware implementation. The servers can also be several times clustered to provide fault tolerance. Publish-subscribe paradigm in messaging middleware is a powerful and scalable abstraction, but an appropriate distributed infrastructure is the key to a scalable implementation of a messaging middleware system. However, scalability might contradict with other required properties. Increasing reliability guarantees the involvement of overheads such as logging or detecting retransmitting and missing messages. The amount of traffic resulting from the acknowledgments generated by such protocols is considerable and limits their scalability. Among distributed messaging middleware systems, probabilistic protocols have recently received increasing attention since they fit the decoupled and peer-to-peer natures. Instead of providing complete reliability by deterministic approaches, probabilistic protocols ensure that a given notification will reach a given subscriber with a high and quantifiable probability. Appropriate trade-off has to be defined to cope with scalability and expressiveness in messaging middleware systems. In mobile environments, messaging middleware systems should maintain communication during periods of disconnection. A disconnected period might be caused intentionally by the consumers, or by device movements. Mobility support requires recovering messages during the disconnected period, and supporting mobility and disconnected operation should be considered as a part of the reliability support. An intelligent caching helps to reduce communication over unreliable links and indirectly contributes to obtain high reliability. Content-based subscription gives flexibility through a combination of topic-based subscription to construct reliable and efficient message flow in distributed messaging systems. It limits the network traffic to unnecessary consumers. Pronto aims to accomplish high reliability for mobile applications by applying and comparing different techniques. Section 2 gives an overview of Pronto and deployments. More details on Pronto functions follow in Sections 3-6. This paper finishes with an overview of related work (Section 7) and a conclusion and future work (Section 8). See example applications and benchmarks for performance improvement in [18], [20].

2 Pronto Overview

Pronto is designed as a middleware forming a collection of distributed services. Three main functions form the backbone:

MobileJMS Client: MobileJMS Client is designed to adapt the JMS client to mobile environments. The challenge is to accommodate JMS with constrained mobile devices. The specifications and interfaces of JMS are complex, but not all functions are mandatory for mobile environments. For example, Pronto does implement neither map-message type nor message priority. On the other hand, as an extension of the JMS API, Message Selector is added to support content-based subscription. It is important to limit incoming network traffic to mobile environment by content-based subscription, that provides more precise message filtering function. One of the aims is to create a mobile-specific JMS client API to optimize slim client library to obtain high reliability. MobileJMS Client is described in more detail in Section 3. Furthermore, a simple JMS server was implemented to support the MobileJMS Client, which is out of scope of this paper.

Serverless JMS: Serverless JMS is a novel serverless version of MobileJMS Client. The aim is to put the JMS scheme in a decentralized model, using IP multicast as transport mechanism. In mobile environments, the nature of data distribution may often fit better into a multicast/broadcast model. Multicast transport mechanisms allow the efficient transmission of data from one to many without redundant network traffic. The basic service provided by IP multicast is an unreliable datagram multicast service, and there is no guarantee that a given packet has reached all intended recipients of a multicast group. Serverless JMS implements both reliable and unreliable multicast transports. A novel protocol to support reliable multicast is designed using a negative acknowledgment. Serverless JMS will perform best over an ad-hoc network. The ad-hoc network is a dynamically re-configurable network without fixed infrastructure and without a requirement for the intervention of a centralized access point. The message domain publish-subscribe in JMS can reside on an ad-hoc network, but not implementations based on a centralized server model. Here, Serverless JMS will play an important role. Serverless JMS can be deployed over a high-speed bus for transmitting a large number of messages distributing the workload of one to several servers.

Gateway and SmartCaching: An intelligent Gateway [18] is introduced as a message hub with store-and-forward messaging, taking advantage of plug-in components for caching, device specific transport, and message transformation. Multiple gateways can be deployed to construct a message broker network over JMS servers, which should produce a reliable and efficient message flow from the source to the consumer. Store-and-forward messaging gives distributed cache and it provides reliability option even it causes messaging process redundant. SmartCaching is designed to provide generic caching for N-tier architecture; it

is embedded as a central function for message storage in Gateway. Gateway and SmartCaching are key technologies for improving messaging among mixed mobile-tier environments in dynamic connectivity scenarios (see Sections 5-6 for details). Plug-in components are not discussed in this paper.

2.1 Distributed Systems with Pronto in a Centralized Model

Figure 1 shows an overview of a distributed system with Pronto in a centralized model. Different deployment possibilities are illustrated:

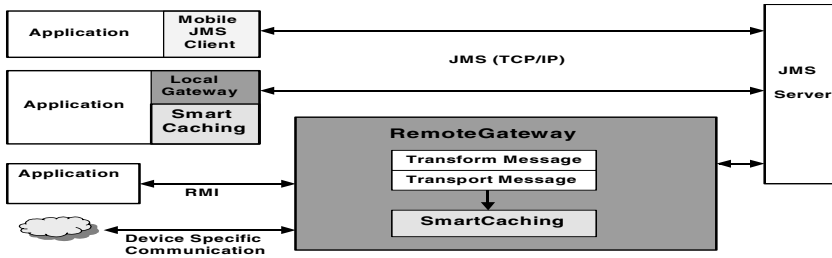


Fig. 1. System Overview of Pronto in a Centralized Model

- **Application with MobileJMS Client:** An application in a mobile device uses a MobileJMS Client API; it communicates directly with the JMS server.
- **Application with LocalGateway:** An application in a mobile device uses a Gateway API. LocalGateway is a mode of Gateway and can run as a separated thread from the application or within the application and performs caching and transcoding through plugged-in components.
- **Application with RemoteGateway:** An application in a mobile device uses a Gateway API. RemoteGateway is another mode of Gateway and runs as a separate process. Currently RMI-based transport between a RemoteGateway and MobileJMS Client is implemented.
- **Non-Programmable Devices with RemoteGateway:** Non-programmable devices require RemoteGateway to perform proper transportation and message transformation for the target devices. RemoteGateway represents every subscriber and publisher for the non-programmable device.

2.2 Pronto in a Decentralized Model

Serverless JMS supports a decentralized model. A publisher acts as a temporary server and keeps a subscription list. Serverless JMS performs the best over ad-hoc network environments (Figure 2a), and over high speed bus (Figure 2b). In Figure 2b, JMS BUS is a Serverless JMS over a high speed bus, which supports to distribute a server's work load to several servers, or replicate a large amount of data to backup machines with classifying to the topics. Serverless JMS can utilize auto-discovery function to maintain subscriptions. Current serverless JMS requires IP Multicast capable routers.

2.3 Distributed Gateways over JMS Servers

Multiple Gateways can be used to distribute JMS messages to the target Gateways where messages are sent to the devices. Gateways are like message brokers, and they can form arbitrary topologies among themselves. Serverless JMS can be also embedded in Gateway as a plug-in component and the cascaded gateways act as multi-message hubs for distributing messages. JMS BUS is a Serverless JMS over a high-speed bus. A high-speed bus can be LAN-based or WAN-based as far as the routers allow IP multicast. Combination of Gateway and JMS BUS offers powerful message flow control for optimization and reliability support, as appropriate for the network characteristics. A deployment example is shown in Figure 3. When the publisher publishes video message, the first gateway persists it in cache. The message is transformed to audio data by extracting the audio portion from video data then sent to the gateway subscribers. In this scenario, all the gateway subscribers are supposed to distribute messages either audio or short text. The gateway can be residing in the mobile device, and it can move from one network to another network as far as the client identifier is set uniquely within the name space. Gateways can act like event brokers grid to distribute messages in reliable and efficient manner. The principle is with increasing distance from the source, data are expected to become more localized by deploying Gateways.

3 Mobile JMS Client

MobileJMS Client is designed to follow the JMS API model. The common building block of a messaging service is the message. Messages consist of events, requests and replies that are created by and delivered to clients. Messaging services such as persistent delivery, durable subscription, the time-to-live option on a message, and transactions show the range of delivery methods. Durable subscription is essential to support disconnected operation in Pronto. Asynchronous messaging is a key integration point in mobile environments. JMS defines two messaging paradigms, publish-subscribe and point-to-point, the latter being less suited for mobile environments. Pronto implements

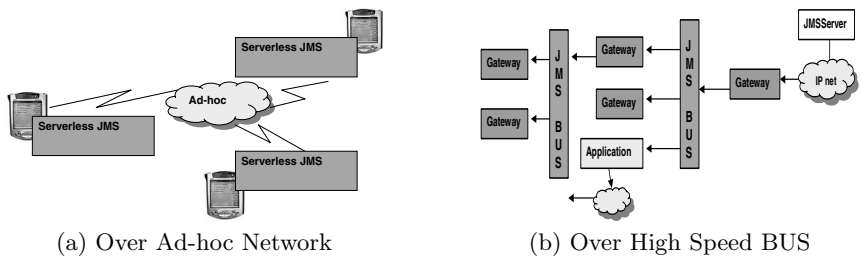


Fig. 2. Pronto in a Decentralized Model

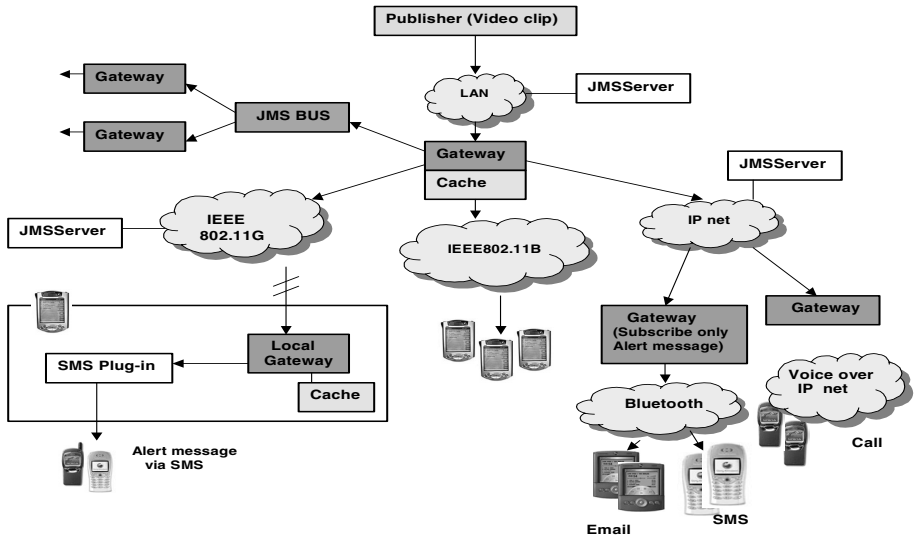


Fig. 3. Multiple Gateways Deployment

publish-subscriber paradigm. JMS does not define standard address syntax, using instead the *Destination* object that encapsulates the address [4]. In the publish-subscribe paradigm, the *Destination* is called Topic. Messages are sent to the *Destination* rather than specific processors or ports. Communication is typically one-to-many and asynchronous. The publish-subscribe paradigm supports the development of location-independent applications that can be moved from one machine to another without affecting their peer applications. In order to specialize JMS to a mobile environment the points below are considered:

Connection: A connection represents an open connection to the JMS server. JMS does not define any specific transport mechanism. In Pronto, HTTP via TCP/IP is implemented, which allows the applets using MobileJMS Client to connect through firewalls to the JMS server. SSL pipe can be used to provide secure connection.

Session: Connections create Sessions. A session is a single-threaded context that handles message-passing operations. A JMS server has a session pool and can execute separate messages concurrently, thus improving performance. If client code needs to receive asynchronous messages concurrently, the client can use multiple sessions to partition client operations, i.e. one client thread can drive one session while another client thread can drive another. Each session object serializes execution of message listeners. Thus, message listeners can share session-related resources. In order to avoid the complex threaded model, connections and sessions share one thread to receive a message in Pronto.

Message Flow: Figure 4 and 5 show the differences between two delivery modes and figures explain the message flow including JMS server behavior. When using the durable delivery mode, each message is stored by the JMS server persistently before delivery to the consumer and is removed after message delivery. This has a huge impact on performance but gaining reliability.

Figure 4 shows the durable subscription/persistent mode of the message flow. The message has an expiration time, which is from the time-to-live beyond the time of publication. It can be set to 'forever' but will be discarded as soon as delivery to all current subscribers and all durable subscribers is complete, as recognized by the acknowledgments of the subscribers. If message delivery 'persistence' is specified, the JMS server holds the message in persistent storage. Figure 5 shows the non-durable/non-persistent mode of message flow. For non-durable messages, the time that messages take to deliver to the Destination depends on their numbers and Destination sizes.

A large number of messages collected in a Destination will take more time to deliver. Redelivery delay time defines when to redeliver a message if a failure occurs. With shorter times, the frequency of redelivery is high, thus increasing network traffic. High Redelivery delay times give therefore better performance. Redelivery Limit defines the number of times a message should be redelivered. Although the probability of guaranteed messaging is lower with lower Redelivery limit, it increases performance due to reduced memory overhead for non-durable messages and persistent overhead for durable messages. It is therefore important to set a sensible Redelivery limit to balance between high throughput and reliability.

Durable Subscription: A durable subscriber registers a durable subscription with a unique identity. A subsequent subscriber with the same identity resumes the subscription in the state left by the previous subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire. This supports not only disconnected operation but also location independence automatically.

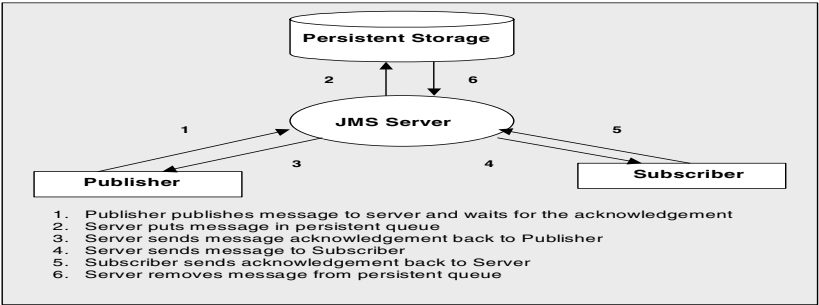


Fig. 4. Persistent Message Delivery

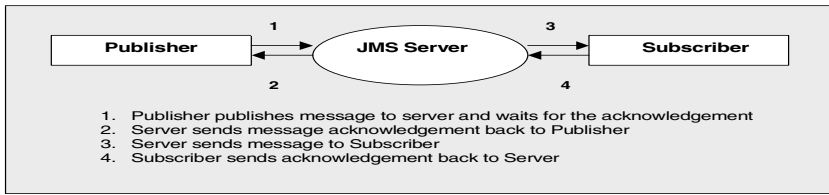


Fig. 5. Non-Persistent Message Delivery

Message Selector (Content-based Subscription): Topics can be structured into hierarchies, and subscriptions can be a part of the hierarchy. This gives content-based messaging and greater flexibility for applications as there is less coupling between producers and consumers. Content-based addressing is more consumer-oriented, whereas subject-based addressing is producer-oriented. Content-based subscription is not part of the JMS specification, but it can be effective in mobile environments to control traffic to gain the reliability. Pronto takes an approach to provide Message Selector for content-based subscription as an extension of JMS API. Message Selector is a filter for a topic defined by the consumer. In Pronto, this filter is implemented with an XML-based TextMessage. A message selector is a string, whose syntax is based on a subset of SQL92 conditional expression syntax.

4 Serverless JMS

Many underlying transmission media such as Ethernet provide support for multicast at the hardware level. Applications with Serverless JMS over such a network lead to a significant performance improvement. Serverless JMS currently implements basic functions, while some JMS features such as persistent delivery and durable subscription were omitted, given the nature of the network model and IP multicast protocol. Key features are shown below:

Multicast Group: Groups of machines representing a multicast group are identified by an IP multicast address. Each address can be considered as a channel to identify groups of hosts interested in receiving the same content. Two channels are used: the ManagementChannel serves administration purposes, while the MessageChannel serves message transmission. As an option, MessageChannel can be defined on each topic.

Reliable Protocol: The basic service provided by IP Multicast is an unreliable datagram multicast service. With such a service, there is no guarantee that a given packet has reached all intended recipients of a multicast group. Serverless JMS implements both reliable and unreliable multicast transport. The reliable version uses a negative acknowledgment. The transparent fragmentation and re-assembly of messages that exceed a UDP datagram size is implemented.

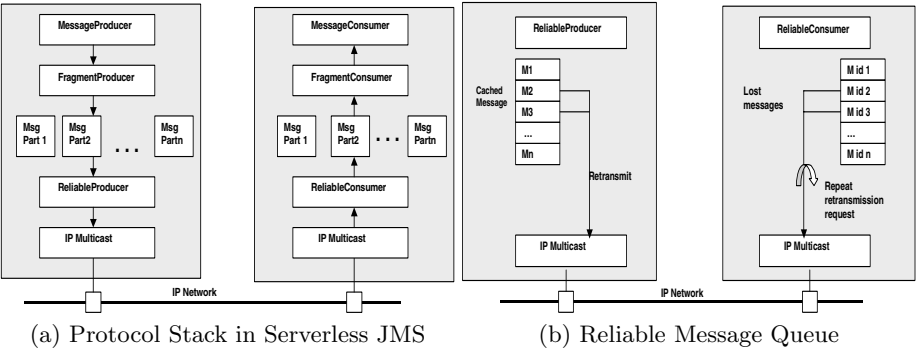


Fig. 6. Reliable Protocol in Serverless JMS

This provides the highest possible delivery guarantee in a multicast environment. Figures 6a and 6b show the components to support the reliability and the fragmentation in Serverless JMS. On the producer side, the *FragmentProducer* splits the message to fit into the packet if the message to be published is larger than the defined network packet size. The *ReliableProducer* assigns a unique order number to each message to allow identification by the consumer. At the consumer side, the *ReliableConsumer* keeps an array that contains the unique IDs of the messages. Once the *ReliableConsumer* detects lost messages, it sends a retransmission request with the missing message IDs. This retransmission request repeats until the messages are retransmitted or it runs into a timeout. The retransmission is performed via the MessageChannel. Once the ordered message comes in, the message will be pushed up to the *FragmentConsumer*. The *FragmentConsumer* reassembles the data and pushes them up to the *MessageConsumer*. The *ReliableProducer* retransmits the requested messages on demand from the *ReliableConsumer*. The *ReliableProducer* keeps the message cache array. This package provides reliability on message delivery. However, this compromises throughput, because it slows down the procedure when the producer interferes with the retransmission process.

Pronto uses deterministic protocols to provide reliability, which retransmits the messages until all recipients acknowledge their receptions or are declared failed. When the network is congested or a receive is perturbed, messages accumulate in buffers until fully acknowledged. Since buffers are bounded, once they fill up the producer is not allowed to send further messages.

Flow Control: The speed of the modern LAN transmission is high, and packet loss will be rare due to good network quality. However, due to the high speed, if the network buffer is not large enough and the subscriber cannot keep up with the speed of incoming data, the buffer will be overwritten and messages will be discarded. It is important to set the network buffer size large enough, but the recent high-speed router fills up the buffer quicker than the subscriber can process. That creates the same symptoms as when packets are lost during transmission. The most effective way to avoid overwriting of the buffer is slowing

down the publishing messages and giving time to the subscriber to keep up with the speed of incoming messages. In *ReliableProducer*, a mechanism slows down or speed up message sending, depending on the detected transmission speed. The *ReliableProducer* keeps a table containing the delay values. Between the sending of two messages, the *ReliableProducer* does wait for the time of these timeout values. The message cache array on the producer and the message ID array on the consumer have to be coordinated with the speed of message transmission. If message loss is higher than the amount of cached messages, the consumer has no way to receive the retransmitting messages. In general, flow control is difficult and expensive. There is no perfect logic to control this. One has frequently to compromise between throughput and reliability.

Subscription Registration: Two subscription modes are defined: the administrated and non-administrated modes. In the non-administrated mode, publishers publish messages independently of the existence of subscribers. Administration mode maintains a subscription list and prevents publishing without subscribers.

Auto Discovery: An auto discovery function is designed. A publisher runs an independent thread for auto discovery, which sends management data that require an echo from subscribers via *ManagementChannel* and maintains the subscription list. Auto discovery repeats this at defined intervals.

5 Gateway

Gateway distributes messages to other Gateways and applications. Multiple gateways can be used as appropriate for the network environment and client characteristics. This allows the construction of a distributed messaging system over JMS servers and offers load sharing and load reduction for good performance, leading to achieve an efficient and reliable message flow. Gateway uses store-and-forwarding messaging paradigm, which is an overhead process and redundant. However, to achieve high reliability, a redundant process may be useful. Gateway is designed as a framework to perform plug-in functions for which two interfaces are defined: **Transport** for mobile device transport and **Transform** for message transformation. The plug-in functions should follow these interface definitions. Gateway initially creates *Transport* and *Transform* objects, according to XML-based configuration data. The Encode-Decode component carries out the message transformation as defined in the configuration. Specific configuration utility is not implemented in Pronto.

Plug-In Components: For the *Transform* interface, caching, compression, and semantic transcoding are good candidates to reduce data size and network traffic. Security (encrypting/decrypting data) functions can also be plugged in. Semantic transcoding offers more than simple data reduction. The information itself is made more abstract (to provide compaction), and the data

should be evaluated whenever necessary. In a mobile environment, a reduction of data size on the network dramatically increases performance, and the concept of semantic transcoding is important. The data are linked to an annotation [11], which can be text corresponding to a video clip, a document summary, or a linguistic description of the content for voice synthesis or image data in grayscale/downsized/low-resolution. For *Transport* interface, device-specific transport for non-programmable devices such as Short Message Service (SMS) or email functions are candidates.

6 SmartCaching

Caching is essential for performance improvement by reducing network traffic and improving latency. The cached data can be raw or processed and stored for reuse, thus avoiding revisiting the source and passing the data through the chain of reformatting and representation. SmartCaching, an intelligent cache function, supports multi-tiered applications across platforms and devices. It currently implements basic functions, while persistent caching, cache validation, synchronization, and coherency management are beyond the scope of this study. In SmartCaching, cached data are decoupled from the data source, and cached data can be made active or up-to-date by *CacheHandler*, which is responsible for updating the cache. For example, Gateway is a *CacheHandler*, and it uses SmartCaching to store subscribed messages. Key functions to clients are the **Pull**, **Subscribe** and **Snapshot** services. The Subscribe service provides asynchronous notification of cached data to client applications, and applications do not need to request to pull data that have already been requested. Using the Subscribe service client applications may be event-driven and active. This simple change has a major impact on performance and on the design of the applications. Snapshot provides a specified period that can be used by the mobile application to obtain the last cache image after disconnection. *CacheManager* is the main component in SmartCaching. It creates objects and manages requests and responses to requesters. Cache is an object that contains a key and the actual caching object, kept as a linked list. The *Cache* object contains the expiration date, and the *CacheManager* will remove expired objects. Alternatively, the *Cache* object can be removed once it is delivered to the subscriber. The three main functions above operate in response to requests from *CacheManager*.

Pull: An application requests a cache synchronously.

Subscribe: An application requests a cache update notification to a cache handler, which notifies the application after the cache is updated.

Snapshot: When data are delivered piecemeal to applications in a time sequence, clients should be able to reconstruct the latest view of the information. This can be achieved by obtaining all data from the data source or by retaining

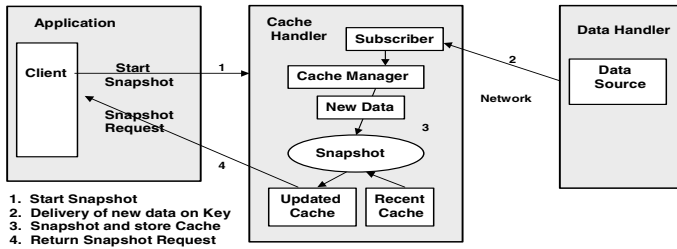


Fig. 7. SmartCaching: Snapshot

the last image in a shared cache. The second option corresponds to the Snapshot service. If the data source sends messages via minimal delta information, caching updates existing data, applying only the delta information. Snapshot needs to know when the baseline starts. Each time a new message is received, the Snapshot rule is applied and the data persist in the cache. The Snapshot rules can be provided by applications. If a client requests Snapshot, it will receive the latest data only. It is the responsibility of the client application that made the Snapshot request to retain all data, and, after the snapshot's arrival, to apply the data to bring that snapshot up-to-date. During disconnection, the client is able to continue to operate using its own local cache to satisfy requests as far as possible. After restoring communication, only the last image of the cache needs to be updated. This can reduce the need for reconnection by skipping all intermediate data. The event notification mechanism can then inform applications of later changes in the underlying cached data. When Snapshot is on, cache update notification is done only when the last image changes. The data flow of Snapshot is shown in Figure 7.

7 Related Work

Reliability specific related work in messaging systems is discussed in Section 1. In this section, related work to Pronto is described. Since the initial JMS specification was released [15], the existing MOM software has been rapidly integrated under the JMS API. Examples are IBM's MQSeries [8], TIBCO's TIB/Rendezvous [17], Softwired's iBus [14], and BEA's WebLogic [1]. However, Softwired's iBus Mobile is essentially the only one to extend JMS to mobile-tier, and designed as an extension of J2EE application servers. It includes a messaging middleware client library compatible with the JMS standard as well as a middleware gateway used to connect mobile applications to J2EE application servers. It supports mobile communication-specific protocols such as GPRS and UMTS. In contrast, Gateway in Pronto is a message hub that can reside in the device or anywhere in between. Pronto provides a flexible N-tier layout, deploying multiple gateways instead of a tight linkage with a server. Gateway offers more than a transport protocol as described above. IBM's MQSeries Everyplace

belongs to the MQSeries family of business quality messaging products. It is designed to satisfy the messaging needs of lightweight devices with own API. There is no standard messaging API for the mobile environment.

The original iBus before JMS heavily used multicast. Currently, several JMS products support multicast transport such as TIB/Rendezvous. However, JMS has not been tried on mobile ad-hoc networks. Much research currently focuses on general datagram routing in both unicast and multicast routing [13,9], but no definite solution to provide JMS semantics using these protocols has been provided. An example of a drawback of using multicast is the drastic performance reduction with redundant rebroadcasts [12]. For reliable transport over IP multicast, various protocols such as SRM [5], RMTP [10], and TRAM [3] are implemented. Pragmatic General Multicast (PGM) [6] is a reliable multicast support protocol for applications that require ordered or unordered duplicate-free multicast data delivery from multicast sources to multiple receivers. For publish-subscribe messaging systems, PGM provides a building block for the messaging system itself, allowing higher performance and scalability for messages that need to go to many destinations. This is a promising approach, but the PGM header is not yet supported by any Java package. For now a reliable protocol based on negative acknowledgment is designed and implemented in Pronto.

Optimizing data over a wireless environment has been successful, although most technologies are tightly coupled with the applications or the servers, based on a client-server model. Techniques for optimization include caching, protocol reduction, header reduction, and adding an asynchronous model. For example, IBM's WebExpress [7] provides a web browser proxy between mobile clients and a web server to optimize HTTP data. Caching is also tied to applications in most cases. Java Temporary Cache (JCache) [16] has been proposed (but not yet implemented practically) by Oracle and provides a standard set of APIs and semantics that are the basis for most caching behavior including N-tier support.

8 Conclusion and Future Work

Pronto attempts to integrate technologies into a compact semantics-based middleware in support of issues specific to mobile environments. A mobile computing environment is resource constrained and support of reliability requires precise functionalities in both explicit and implicit ways. Deploying different plug-in functions with Gateway demonstrates the construction of a reliable and efficient message flow over a publish-subscribe system. Using disconnected operation and SmartCaching improves flexibility for the design of mobile applications.

JMS is more complex than discussed here. Although JMS lacks security definition, Pronto would need to support other reliability functions such as administration, security, error handling and recovery as well as distributed transactions. Most importantly, it is critical to establish a standard API for publishing, managing, and accessing public references to distribute functionality over mobile environments; this includes security aspects such as encryption, authentication, and access control on distributed objects.

Acknowledgment. I would like to thank Jean Bacon and Jon Crowcroft (University of Cambridge) for critical reading and constructive comments.

References

1. BEA. WebLogic 6.0 JMS. <http://www.bea.com>.
2. K. H. Britton et al. Transcoding: Extending e-business to new environments. *IBM System Journal*, 40(1), 2001.
3. D. Chiu, S.Hurst, M. Kadansky, and J. Wesley. TRAM: Tree-based Reliable Multicast Protocol. *Sun Microsystems Technical Report TR-98-66*, 1998.
4. P. Eugster et al. The Many Faces of Publish/Subscribe. *Technical Report TR-DSC-2001-04, EPFL*, 2001.
5. S. Floyd et al. A Reliable Multicast Framework for Light-weight Session and Application Framing. *ACM SIGGOMM Communications Review*, 1995.
6. J. Gemmell et al. The PGM Reliable Multicast Protocol. *IEEE Network special issue on Multicast: An Enabling Technology*, 2003.
7. B. Housel and D. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. *Proc. of Int. Conf. on MobiCom*, 1996.
8. IBM. MQ Series. <http://www.ibm.com/software/ts/mqseries/>.
9. S. Lee et al. On-Demand Multicast Routing Protocol. In *Proc. of WCNC*, 1999.
10. J. Lin and S. Paul. Reliable Multicast Transport Protocol (RMTP). *Proc. of IEEE INFOCOM '96*, 1996.
11. K. Nagao. Semantic Transcoding: Making the World Wide Web More Understandable and Usable with External Annotations. *Proc. of Int. Conf. on Advanced in Infrastructure for Electronic Business, and Education on the Internet*, 2000.
12. S. Ni et al. The broadcast problem in a mobile ad-hoc network. In *Proc. of ACM/IEEE MobiCom*, 1999.
13. E. Royer and C. Perkins. *Multicast Ad-Hoc On-Demand Distance Vector Routing*, 2000. draft-ietf-manet-maodv-00.txt.
14. Softwired. iBus Messaging . <http://www.softwired-inc.com/>.
15. Sun Microsystems. *Java Message Service (JMS) API Specification*. <http://java.sun.com/products/jms/>.
16. Sun Microsystems. *JCache: Java Temporary Caching API*. <http://www.jcl.org/jsr/detail/107.prt>.
17. TIBCO. TIB/Rendezvous Concepts . <http://www.rv.tibco.com>.
18. E. Yoneki and J. Bacon. Gateway: a Message Hub with Store-and-forward Messaging in Mobile Networks. *Proc. of ICDCS Workshops MCM*, May 2003.
19. E. Yoneki. Pronto: Messaging Middleware over Wireless Networks. *4th ACM/IFIP/USENIX Int. Conf. of Middleware(Work in Progress)*, 2003.
20. E. Yoneki. Mobile Applications with a Middleware in Publish-Subscribe Paradigm. *Proc. of the Third Workshop on ASWN* , 2003.

A Flexible Middleware Platform with Piped Workflow

Wanjun Huang, Uwe Roth, and Christoph Meinel

Department of Computer Science, University of Trier
D-54286 Trier, Germany
{huang,roth,meinel}@ti.uni-trier.de

Abstract. Middleware emerges as an excellent solution for the complicated distributed computing application. But as the appearance of new devices and new applications, the inflexibility of traditional middleware system becomes more serious and urgent. In this paper we propose a new flexible middleware platform which adopts the technologies of piped workflow and computational modules to provide a modular and extensible platform for future applications. The piped workflow provides a very flexible mechanism to organize all computational modules working together. During the running time, all computational modules communicate only with data channel of the piped workflow through which they can keep extremely independent, and the flexible flow control strategy makes the application programmer convenient to arrange all functional components for variable customer requirements.

1 Introduction

The emergence of middleware has solved a serial of problems arose by applications widely distributed on network, and evolves the distributed computing model from client-server into three tiers architecture. As the middle tier component between the low level operation system and the top application software, middleware helps programmer easily and quickly build distributed business application without considering some common but complicated problems, such as heterogeneity of operation system, complexity of communication, concurrent interoperability, system stability, transfer security and so on. Almost all traditional middleware solutions adopt the mechanism of block box and have gained big success in providing remote procedure access. However, as appearance of new devices and new applications, the inflexibility and limitations of traditional middleware become more serious and absorb many researchers' interests and concerns. When network just comes into our life, how to easily build a distributed application is the first task. But now, as network become more pervasive, and as portable handheld devices become more popular, the more important problems become how to make the middleware more extensible, reusable and adaptive for unknown future applications and how to build middleware itself more easily. In order to provide more convenience for the application writer, the middleware has to undertake more functionalities and responsibility, which make it hard to

complete the implementation of a fat middleware server. Additionally, modularity and extensibility are also another two big unsolved problems. The traditional middleware are composed of fixed components with fixed policies that can't meet the diversity needs of varying applications environment. Although some solutions [4], [5], [8] have been proposed to dynamically customize the middleware components to adapt the variety of client environments, these varieties can be predicted and have been considered during the design of original architecture. For every unknown or unpredictable application, some original source codes have to be modified to allow the new processing components working well along with old ones. For example, consider an e-business application running on a distributed environment. Initially the transactions processing are some common remote data access and remote procedure invocation, so the traditional middleware solution is capable to deal with them. Subsequently, the middleware server is required to provide multimedia services, so stream transfer protocol, MPEG encode/decode and quality of service have to be added into inner key components. Eventually, company's business services want to be expanded to mobile commerce, so some filter components are also needed for mobile computing. If all these changes will inevitably arise to the modification of source code or even the structure of system, then each update of middleware will bring big price of time consuming and hard work. So it's necessary to explore and research more flexible middleware architecture to improve the efficiency of system implementation and make the key components of middleware more independent to adapt for the future update.

Component oriented software development, such as Java Bean and ActiveX Control, support the construction of sophisticated system by assembling a collection of components software with the help of visual tool or programmatic interfaces. However, they have to integrate in the level of source code and support little for dependences management, which are important for component to be freely loaded into or unloaded from inner workflow to work fluently in distributed system. Here we propose a new flexible middleware platform, which adopt a piped workflow and computational module to achieve the integration of independence and the flexibility for future extension. In the following, firstly we will introduce the architecture of new middleware platform and explain some important components. Subsequently, the computational module and piped workflow will be described in details. And then we discuss the related work recently proposed by other researcher. At last conclusion will be made to summarize its features and outline our future work.

2 Architecture

To testify the new ideal, we have designed and implemented middleware platform - Smart Data Server V2.0 (SDS 2), which has adopted the technologies of computational modules and piped workflow. All components can fall into three catalogues: infrastructure, computational modules and application services. The infrastructure is laid on the bottom level that contains piped workflow and some core services. Computational modules live only inside of piped workflow and are

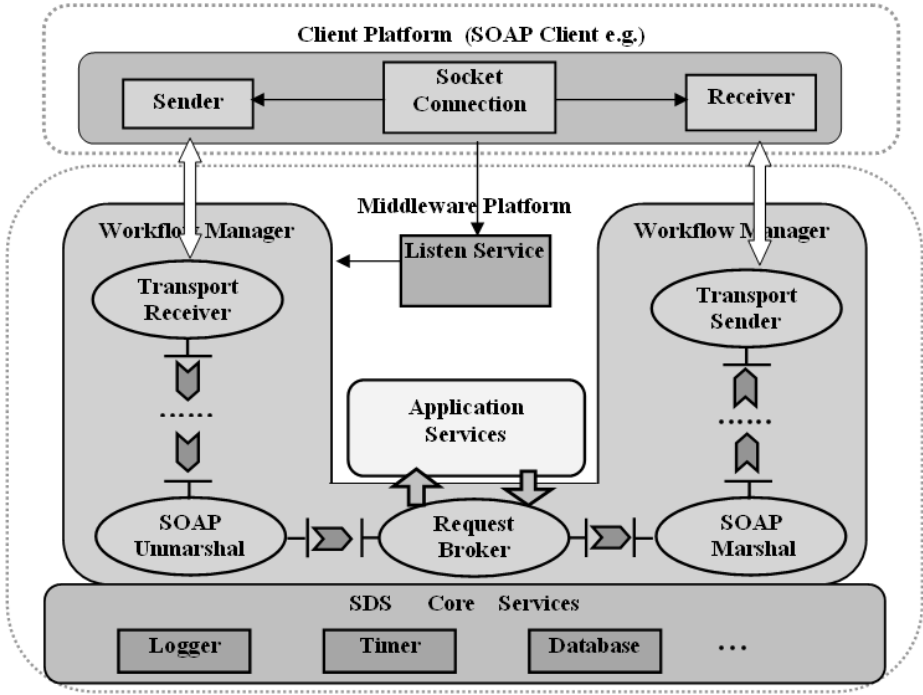


Fig. 1. Piped workflow middleware architecture

the primary components to process client request. The relation diagram of all these components is depicted as figure 1.

2.1 Core Services

The Core Services provides some fundamental services that will be used in the whole middleware platform. Not only in the workflow manager and computational modules, these services can also be used directly in the application services. SDS 2 Core Services include Logger, Timer and Database etc which provide consistent services using the technology of thread instead of Application Programming Interface (API). Logger Service is used to record error, warning, debug and running state information which are convenient for programmer to debug the system and for administrator to check the middleware server. Timer service can activate some task at one specific time or periodically that is very useful for the some special application, such as checking email etc. Listen service is also an important part of Core Services, and its responsibility are to listen socket request and establish socket connection with client that will be transferred to workflow manager to activate a workflow processing routine.

2.2 Workflow Manager and Modules

Workflow Manager provides the service for piped workflow to organize all relevant modules working together. Here the implementations of computational modules contain TransportReceiver, SOAPUnmarshal, RequestBroker, SOAPMarshal and TransportSender. In our implementation, we also finish another two modules, namely IPTPUnmarshal and IPTPMarshal, to transfer the request and response messages using self-defining protocol. - Information Package Transfer Protocol[1]. So, two transfer protocols are available at the same time according different flow solutions.

2.3 Application Service and Client Tools

Application service is an application repository where store the current services provided for client. These services can be accessed directly via Java reflection technology. So there is no requirement for user to declare the interface of deployed services, but the price for this flexibility is the limitation of usage of Java Reflection. The client tool depends on the transfer protocol. One of our adopted protocols is Simple Object Access Protocol (SOAP). It's an international standard protocol, so any client package developed by companies or open organizations can be used as our client tools, such as Apache SOAP client package, Microsoft SOAP toolkit client package etc.

3 Computational Module

Component technology has showed its power in the development of sophisticated software system, such as Java Bean and ActiveX Control. In the middleware system, component should not only keep its characteristics of inner integration and independence, but also hold a high ability to be flexibly controlled and integrated into the existed system. At present component technology has already been applied into most of current middleware systems or products, but different components are always mixed into each other in some extent when they work together. Here the proposed components structure calls computational modules that can avoid these confused problems. As described in figure 2, computational module exhibits its outer behaviours in the form of following four items:

- *Input Interface* that represents what computational module need for internal processing.
- *Output Interface* that shows the processing result of module.
- *Properties* that describe initial setting for module.
- *Requirements* that list the basic conditions of module that are used to verify inputted data.

Module exhibits its IO behaviors through input and output interfaces and hides all the inner implementation detail as a black box for application writer. To reify the properties, module contain a serial of basic methods, such as methods

for initialization and release, to ensure it can be load and unload in any time. Requirements are set to control whether inputted data are qualified. This step is benefit to optimize processing, and also very crucial for workflow manager to decide which is the next module after current one. The construction of computational module is based on the technology of thread, which make it able to run independently. During the reification of input and output interface, all modules have to implement a behavior interface, which inherit from a common IO interface to generate the common IO behaviors. Workflow manager can access all IO behaviors of every module via the way of looking up member methods of its behavior interface. Input interface tells workflow manager what it needs, and output interface tells what it will produce. All the methods of input interface will be executed by piped workflow to get the required data from data channel before the computational module runs, and it will also produce its results into data channel after finishing of running. So speaking strictly, all modules keep no contact with each other and they just communicate with data channel of workflow manager. This independent property of computational module brings much flexibility to add a new one into the system.

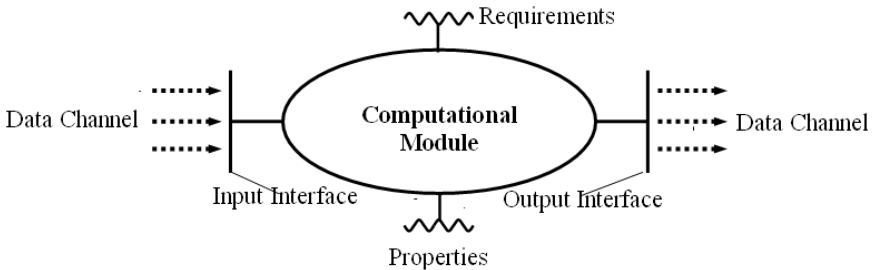


Fig. 2. Structure of computational module

During the procedure of implementation, programmer can develop many modules according the actual requirements. But not every module has to be engaged in the processing of one request, and it depends on the workflow solutions and client request to decide which module will be used for one processing. In future more modules can also be freely added here to adapt the middleware for new application, such as MPEG encode and decode modules for multimedia application etc.

4 Piped Workflow

In traditional middleware systems inner workflow is always integrated closely with processing components which result in the inextensibility for future extension. To achieve more flexibility for future components and unknown requirement, we propose a piped workflow to adapt this variation. The piped workflow consists of Workflow Nodes, a Data Channel and a Flow Control.

4.1 Workflow Nodes

There are three kinds of workflow nodes: Program-Nodes, Flow-Nodes and End-Nodes which are all inherited from a same parent node. Program-Node is a special node used in workflow program. It is not only tied with a start module where the flow control starts, but also used to control other modules and specify sub-routines. Flow-Node is the core node inside of workflow. It is tied with most computational modules, which take charge of the primary work of request processing. End-Node is tied with last processing module. In fact, all these three kinds of nodes will not do any real work for request processing, and they are just used to control the workflow whose primary task is to decide which module is the next one. The actual works are done by computational modules that are associated with workflow nodes through the implementation of pipe interfaces, as depicted in figure 3. Using the corresponding relation between computational modules and workflow nodes, piped workflow mechanism can flexibly control the computational modules through the control of workflow nodes. When a new module is added, what the administrator needs to do is to bind the new module with a specific workflow node in the configuration file.

4.2 Data Channel

The traditional middleware components can not keep absolutely independent because they have to directly communicate and exchange data each other. In piped workflow a special media - data channel is provided to exchange data among modules. Here there are two ways to realize its functionality of data exchange. One is data pool, and another is pipe, as described in figure 4.

Data Pool. Data pool is similar with the conception of bus in area of Integrated Circuit (IC). It is a common media where the communications between

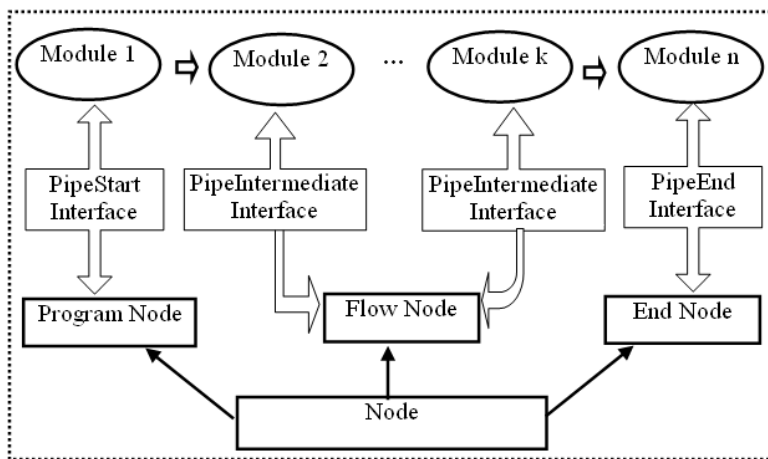


Fig. 3. Relation between workflow nodes and computational modules

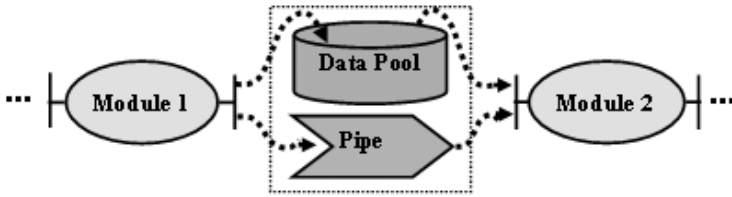


Fig. 4. Data channel

different components are available. Data pool provides services for all relevant modules during one processing procedure. It will be generated when a processing of one request starts and destroyed when the processing of request finishes. The implementation of data pool contains a public temporary storage and communication ports between data pool and modules. Each module has to define which data it needs from data pool and which data will be produced to data pool. These functionalities are realized through the implementation of an IO behavior interface extended from a common interface. In each I/O-behavior interface the set- and get-methods are defined to represent the input and output behaviors respectively. Under the management of workflow manager these variable data are exchanged between the modules and data pool through Java reflection technology. Each set-method of the module will be checked for its associated input data after the module is initialized and each get-method will also be checked to get result after execution of the module. If later the module is replaced by another different module, the interface behavior of new module should not be changed. This strategy ensures that other modules still can communicate with such an old module whose entity has been changed.

Pipe. Data pool act as the role of software buses where multi-modules can take the same output of another module as their common source data. It is very convenient for data communication, but can not offer any dependencies information between adjacent modules. So pipe is proposed to link relevant modules and reveal the dependences of them. Through the way of pipe control messages can be transferred to manage the operation and dependency of each module. Pipe consists of several independent sub-pipes that serve only for two adjacent modules. Each sub-pipe contains a pair of ObjectIO that includes ObjectIn and ObjectOut. ObjectIn is used for previous module to transfer data from module to pipe, and ObjectOut is employed for latter module to transfer data from pipe to module. In one pair of ObjectIO, ObjectIn and ObjectOut share the same temporary memory, so the dependency messages can be transferred from the first module to last one via a serial of sub-pipes. Additionally, different type of computational module contains different composition of ObjectIO. For example, a modules implementing PipeIntemedia Interface contains both ObjectIn and ObjectOut. But the computational module implementing PipeStart Interface or PipeEnd Interface only contains ObjectOut or ObjectIn respectively. Using such a pipe mechanism, the dependencies of different modules are represented and control messages can be transferred automatically and orderly.

4.3 Flow Control

The flow control is a control strategy for all nodes of the workflow program. It decides which node will be processed after the current one. Here two factors result in the flexibility of flow control. One is the structure of computational module and data pool that can arrange the node freely according requirements. Another factor is the dynamical control strategy. It means the result of the flow control is not a fixed link of all nodes and will vary dynamically depending on the flow solution and the execution state of each node.

To manage the workflow nodes, different flow solutions should be configured in the configuration file in advance. Just as figure 5, two flow solutions are illustrated where one solution comprises nodes linked by red arrowhead and another one is represented through nodes linked by grey arrowhead. In a valid solution there should be only one module set to start-node from which the processing work start and at least one module directly head to end-node. For a selected flow solution, such as one indicated by red arrowhead, it may also have different routes during the running time. The real arrowheads indicate the normal default route, and dashed arrowheads indicate other possible routes. When a module is running, it will first check the requirements according input-ted data, and then return an execution state with which workflow manager can decide next route. For example, in the red arrowhead solution of figure 5, the node "A" will go to node "B" if it returns a normal execution state code "1", and it will go to node "D" if the returned execution state code is "2". Because computational module runs as an independent thread, acquirement of execution state does not mean the end of module execution. The execution state is just to indicate the processing response of current module for next module according the inputted data and requirement of current module.

5 Related Work

Component oriented software development has become the popular accepted tendency for reusable and extensible distributed application. But how to represent the dependencies of different components and how to organize these components

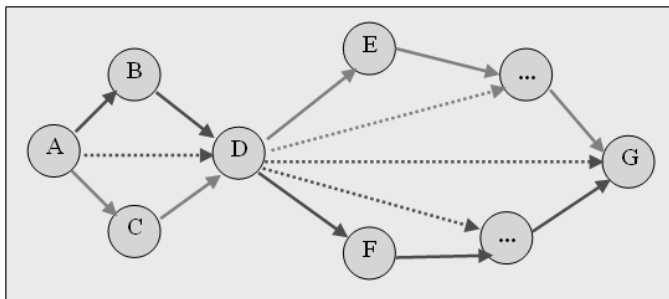


Fig. 5. Paradigm of flow chart

still remain many problems. In [6], authors proposed two distinct kinds of dependences to manage the dependences of components in distributed system. One is prerequisite that are the requirements for loading an inert component into the running system. Another is dynamic dependency that is used to manage the loaded components in a running system. To reify the dynamic dependencies, Kon et al. design a Component Configurator that is responsible for storing the runtime dependencies between a specific component and application components and other system. Each component *C* has a set of hooks to which other components can attach. These hooked components are the components on which *C* depends. There might be some other components (called clients) that depend on *C*. Through the communication and event contact between each component with hooked components and its client, dynamic reconfiguration is enabled for components that are already running. But Component Configurator just define the dependences of component, it has no consideration of the communication between different components. So the component has also to define the same specific behavior interfaces with that of alternative components, just like traditional way. This limitation also restricts the flexibility degree of dynamic reconfiguration. G.S.Blair et al. [3], [4] proposed a configurable and open middleware platform based on the concept of reflection. In their reflective middleware, they introduced open binding and component framework to support the construction of meta-space. Component framework consists of primitive components, which include different kinds of basic and indivisible functional unit, and composite components that represent the configuration of primitive components and composite components. Inside of open binding the communication of different components is realized through the implementation of local binding that is pity to have not been clarified the implementation details. The rather similar idea can be found in [10], Shrivastava present a workflow based model for distributed applications. In their model, workflow schema is used to represent the structure of tasks in a distributed application with respect to task composition and inter-task dependencies. Task controller is used to guide the workflow execution, just like our workflow nodes. But they have not explained clearly and in detailed the conception of computational module and data channel.

6 Conclusion and Future Work

Middleware masks the problem of building distributed application among heterogeneous environment. But the complexity of distributed network and new coming requirement make the construction of middleware more difficult that bring an urgent requirement for high reusability, modularity and extensibility. In this paper we present technologies of piped workflow and computational modules to provide an extensible, configurable and flexible mechanism for the construction of middleware architecture. The computational module exhibits explicitly the properties, requirements and its IO behaviors. Under the management of piped workflow communication between different modules are available and can keep absolutely independent in the level of source code. Additionally, piped workflow

provides an extreme flexible mechanism to schedule all the modules working together according a variable flow control strategy. But in our system flexibility is still not enough. It just provides the ability to arrange modules freely in the server side, and client has no ability to inspect and modify the components structure of middleware server. So how to improve our middleware to gain more ability of reflection is our next research aim. Security is always a hot topic for the field of distributed computing application. Especially when we investigate to provide some inspection and adaptation abilities for client environment, the secure transport, authentication and privilege management have become more important and indispensable to manage middleware platform to meet the need of some crucial and sensitive distributed applications. Additionally, we want to provide some specific services to extend our middleware platform for some specific application area, such as multimedia. But there are still many problems for multimedia supporting, such as stream transfer, quality of service, synchronization and so on.

References

1. U.Roth, E.G.Haffner, T.Engel, Ch.Meinel: The Smart Data Server - A New Kind of Middle Tier. *Proceeding of the IASTED International Conference on Internet and Multimedia Systems and Applications*, 1999, pp. 362–365.
2. Joo C. Seco and Lus Caires: A Basic Model of Typed Components, *Proc. European Conference on Object-Oriented Programming*, Cannes, France 2000.
3. Nikos Parlavantzas, Geoff Coulson, and Gordon Blair: Applying Component Frameworks to Develop Flexible Middleware. *Workshop on Reflective Middleware*, April 7–8, 2000, New York, USA.
4. G. S.Blair, G. Coulson, P. Robin, and M. Papathomas: An Architecture for Next Generation Middleware. In *Proceedings of Middleware'98*, pages 191–206. Springer-Verlag, Sept. 1998.
5. Fabio Kon, Manuel Roman, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhaes, and Roy H. Campbell: Monitoring, Security, and Dynamic Configuration with the dynamic TAO Reflective ORB. *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*. New York. April 3–7, 2000.
6. Fabio Kon and Roy H. Campbell: Dependence Management in Component-Based Distributed Systems, *IEEE Concurrency*, 2000. 8(1): p. 26–36.
7. D.C. Schmidt and C. Cleeland: Applying Patterns to Develop Extensible ORB Middleware, *IEEE Comm. Magazine*, IEEE CS Press. Los Alamitos, Calif., vol. 37, no. 4, 1999, pp. 54–63.
8. Mark Astley, Daniel C. Sturman and Gul A.Agha: Customizable middleware for modular distributed software. *Communications of the ACM*, v.44 n.5, p. 99–107, May 2001.
9. Mark Astley, Gul Agha: Modular Construction and Composition of Distributed Software Architectures. *PDSE 1998*: 2–12.
10. S.K. Shrivastava and S.M. Wheeler: Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications. *The 4th International Conference on Configurable Distributed Systems (CDS'98)*, Annapolis, Maryland, USA, May 4–6 1998.

Using Web Services Technology for Inter-enterprise Integration of Digital Time Stamping

A. Cilardo¹, A. Mazzeo¹, L. Romano¹, G.P. Saggese¹, and G. Cattaneo²

¹ University of Naples Federico II, Via Claudio 21, 80125 Napoli, Italy
{acilardo,saggese, mazzeo,lrom}@unina.it

² Finmatica SpA, Via Amato 15, 84131 Salerno, Italy
g.cattaneo@finmatica.com

Abstract. This paper describes the results of a research activity conducted cooperatively by an academic and an industrial party. It presents a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet for inter-enterprise integration. State-of-the-art time stamping algorithms and crucial issues related to their practical implementation are discussed. Focus is on integration problems which arise when a potentially large community of enterprises – which rely on a handful of heterogeneous technologies – is willing to access remote third-party time stamping services. We propose a practical architecture which provides time stamping services, both in terms of relative temporal authentication, based on a linear linking scheme, and of absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. The architecture has been implemented using the emerging Web Services technology. An integration experiment has been conducted to evaluate the effectiveness of the proposed solution.

1 Introduction

Now more than ever, electronic documents are the heart of businesses life. From accounting spreadsheets to e-mail, from secure filings to the intellectual property of the R&D department, business depends on the integrity of corporate data. In this scenario, a particularly challenging issue is how to guarantee the trustworthiness of electronic records, and in particular how to establish beyond a shadow of a doubt that vital electronic records have not been backdated. In many cases, it might be necessary to do that in a court of law.

In order to guarantee that the trustworthiness of long lived electronic documents can be verified over the years, it is crucial that reliable digital time stamping features be made available.

During the last years, especially in the context of legal regulation of digital signature techniques, the organizational and legal aspects of time stamping itself have become the subject of world-wide attention. Time stamping can be thought

of as a set of cryptographic techniques enabling one to ascertain whether an electronic document existed at a certain time (Absolute Temporal Authentication), or was created before another document (Relative Temporal Authentication).

In the literature, time stamping schemes are classified into three distinct categories [14]: simple, linking, and distributed schemes.

In the *simple* scheme, a time stamp is generated by a trusted third party (the Time Stamping Authority, TSA) in such a way that it does not involve data included in other time stamps. The main weakness of this scheme is that the TSA has to be unconditionally trusted: if the TSA fraudulently alters the time parameter of a certain time stamp, nobody can detect the alteration. Also, if a leakage of the signature key of the TSA has occurred, fake time stamps can be forged at will. Some ten years ago, the only known time stamping methods relied on the use of TSAs which were based on the simple scheme. Thus, applications which needed digital time stamping had no choice but trusting the TSA unconditionally.

In 1991, the seminal publication [11] of Haber and Stornetta showed that the trust to the TSA can be greatly reduced by using the linking schemes, or alternatively the distributed schemes. Several papers were published during the last years, which further improved the original schemes [8,7,12,9]. The basic idea behind the *linking scheme* is to generate a time stamp which involves data included in other time stamps. A chain of time stamps is constructed, typically by using a one-way hash function. If an issuer is willing to deliberately alter or forge a certain time stamp, he¹ has to alter all the related time stamps. For this reason, it is more difficult for an issuer to manipulate a time stamp in the linking scheme than it is in the simple scheme.

Finally, in the *distributed scheme* multiple issuers (who could possibly be the users of the service themselves) independently generate a time stamp according to the simple scheme, each one using his own key and time source. The set of issuers designated to sign the time stamp is chosen randomly, in such a way that the submitter cannot determine it a priori. If the number of signing issuers is less than a specific predetermined number, they cannot produce a correct time stamp. This scheme relies on the difficulty for the submitter to collude with all of the issuers which are needed to complete the stamping process. However, the need for a large number of independent issuers makes the distributed scheme rather unpractical in most real-world scenarios, as compared to the linking scheme.

Although the linking and distributed schemes show that the provision of a trusted time stamping service does not necessarily imply blind trust in the TSA, the actual implementation of a time stamping server is still a hard task, since:

- it uses algorithms and data structures which are extremely sophisticated,
- it requires a complex infrastructure for service delivery to heterogeneous clients, and
- it has challenging requirements in terms of availability and reliability.

¹ Throughout this paper, he and she will be used interchangeably.

For these reasons, having each individual enterprise implement its own time stamping service within the enterprise is not an efficient nor a cost-effective solution.

Neither do major middleware technologies – such as Common Object Request Broker Architecture (CORBA) and .NET – currently provide time stamping services as a built-in facility. Some work has indeed been done on middleware-based time services for distributed systems, but it addressed the issue of time consistency and not of temporal authentication [22,23].

For the above discussed reasons, a promising solution – and currently the only feasible one – is to have a limited number of independent parties, acting as digital notarization agents, provide time stamping services to a large community of users. Surety [21], the American company founded by the two pioneers of time stamping techniques, Haber and Stornetta, has provided such services since 1994. However, this scenario leads to the non trivial problem of how to integrate third-party time stamping services across enterprise boundaries.

This paper presents a web-based architecture for providing time stamping services suitable for an inter-enterprise integration approach. As far as the algorithm is concerned, we adopted a linear linking scheme, since this solution provides the best compromise in terms of security and performance. All time stamping requests falling in a given time window are gathered in a tree-like structure and compressed into a single data item to be signed and linked to the results coming from the previous time windows. The width of the time window represents the time resolution by which relative temporal order between two distinct stamps can be established. Absolute temporal order is achieved by having the time stamping authority add a time value to the time stamp token before the signature phase.

As to the architecture, we resorted to a multi-tier structure. The back-end server is in charge of the bulk of the computing activity. Currently, it is implemented in software. Work is ongoing to implement it in hardware, to boost the performance of the system. The middle-tier is in charge of leveraging the services provided by the back-end servers, in order to satisfy interoperability requirements which arise from the heterogeneity of the service requestors. To implement the middle-tier, we resorted to the emerging Web Services technology, which enables flexible interaction between applications across enterprise boundaries.

The rest of the paper is organized as follows. Section 2 gives an overview of the Web Services architecture. Section 3 briefly describes the linking schemes used to provide time stamping services. Section 4 presents the architecture of the system, and gives some details about its current implementation. Section 5 presents an integration experiment. The experiment is conducted across enterprise boundaries, with respect to a simple case study application. Section 6 concludes the paper with some final remarks.

2 The Web Services Framework

A *Web Service* [15] is to be thought of as an interface describing a collection of operations that are network-accessible through standardized Extensible Markup Language (XML) messaging. It is described using a formal XML notion, called its *service description*. The description covers all the details necessary to interact with the service, including message formats, transport protocols, and service location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented, and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfil a specific task or a set of tasks. They can be used alone or in conjunction with other Web Services to carry out a complex aggregation or a business transaction.

The Web Services architecture is based upon the interactions between three roles: *Service Provider*, *Service Registry*, and *Service Requestor*. The interactions involve the publish, find, and bind operations. Together, these roles and operations act upon the Web Services artifacts: the Web Service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web Service). The service provider defines a service description for the Web Service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry, and uses the service description to bind to the service provider and invoke or interact with the Web Service implementation. Service provider and service requestor roles are logical constructs, and a service can exhibit characteristics of both. Figure 1 illustrates these operations, the components providing them, and their interactions.

The Web Services architecture consists of several layers. To perform the three operations of publish, find and bind in an interoperable manner, there must be a Web Services stack that embraces standards at each level. The foundation of the Web Services stack is the network. Because of its ubiquity, Hyper-Text Transfer

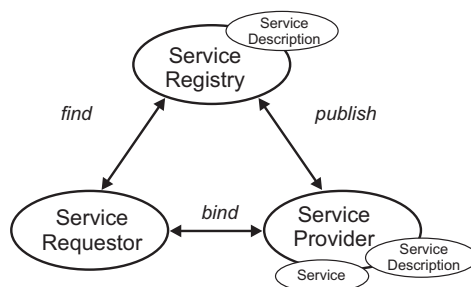


Fig. 1. The Service Oriented Architecture for Web Services.

Protocol (HTTP) is the *de facto* standard network protocol for Internet-available Web Services. Other Internet protocols can be supported, including Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP). The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. Simple Object Access Protocol (SOAP) is the chosen XML messaging protocol. The service description layer is actually a stack of description documents. Web Service Definition Language (WSDL) is the *de facto* standard for XML-based service description. This is the minimum standard service description necessary to support interoperable Web Services. WSDL defines the interface and mechanics of service interaction. Additional description is necessary to specify the business context, qualities of service, and service-to-service relationships. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web Service. For example, business context is described using Universal Description Discovery and Integration (UDDI) data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) document.

Because a Web Service is defined as being network-accessible via SOAP and represented by a service description, the first three layers of this stack are required to provide or use any Web Service. The simplest stack would thus consist of HTTP for the network layer, the SOAP protocol for the XML messaging layer, and WSDL for the service description layer. This is the interoperable base stack that all inter-enterprise or public Web Services should support. Web Services – especially intra-enterprise or private ones – can support additional/alternative network protocols and distributed computing technologies.

3 Linking Time Stamping Schemes

As already mentioned in the introductory section, the *simple* time stamping scheme relies only on a trustworthy third party (the Time Stamping Authority, TSA) which is in charge of certifying the time by signing the stamp. As we have already pointed out, this time stamping system is rather weak, since it relies on an unconditionally trusted third party, which is wholly liable for the issued stamps. Moreover, in the event that the secrecy of the issuer's signature key is compromised, there is no way to distinguish a genuine stamp from a forged one.

Linking schemes overcome these drawbacks. All the proposed time stamping linking schemes realize one-way dependencies by means of the so-called *collision resistant one-way hash functions*. These include families of functions which compresses bit-strings of arbitrary length $*$ to bit-strings of a fixed length l ($h : \{0, 1\}^* \rightarrow \{0, 1\}^l$), and which satisfy the following properties:

- 1 The functions h are easy to compute, and it is easy to pick a member of the family at random.
- 2 It is computationally infeasible, given one of these functions h , to find a *collision* for h , i.e. a pair of distinct strings x, x' such that $h(x) = h(x')$.

The practical importance of such functions has been known for some time, and researchers have used them in a number of schemes. Hash functions have a number of good properties which are well suited for all kinds of time stamping schemes.

In the time stamping linking scheme, hash functions are mainly used to produce time dependencies between issued time stamps, based on the following consideration: if h is a collision-resistant one-way hash function, and the values $h(x)$ and x are known to a supervisor P at a moment t , then someone (possibly P himself) used x to compute $h(x)$ at a moment prior to t . The TSA combines requests from individual clients which arrive within a given time window, along with certain values related to stamps issued in the past. With such a scheme it is hard to produce fake time stamps because forging a single stamp means forging all verifiable dependencies. Moreover, the need for a trusted TSA can be greatly reduced with the linking schemes by periodically publishing the values used to create the dependencies. Actually, if one can demonstrate the dependency of a stamp on some widely accepted data (for example, a hash value weekly published on a printed newspaper), the TSA is no longer involved in the verification process. The audit can be accomplished at any time and allows the verifier to demonstrate not only whether, but also *when* the linking process was altered and thus to distinguish genuine stamps from unreliable ones (with a certain time resolution depending on the publishing rate).

An additional advantage is that one can hash the document x to be time stamped, and submit only the hash value $y = h(x)$ to the time stamping authority. For the purpose of authentication, stamping y is equivalent to stamping x ; however, only a small, fixed-length message is to be submitted to the time stamping authority, greatly reducing the bandwidth problems which would arise if the whole document x were to be processed. Resorting to hash functions solves a privacy issue as well, since the content of the document to be time stamped need not to be revealed to the Time Stamping Service provider. The originator of the document computes the hash values himself, and sends them to the time stamping service. The plain document is only needed for verifying the time stamp. This is very useful for many reasons (like protecting something that one might want to patent). Depending on the design goal for an implementation of time stamping, there may be a single hash function used by everybody, or different hash functions for different users. With a secure signature scheme available, when the time stamping authority receives the hash value, it builds the time stamp token, then signs this response and sends it to the client. By checking the signature, the client is assured that the time stamping authority actually did process the request and that the hash was correctly received.

The most widely known linking schemes are the Haber and Stornetta's ones. The first scheme proposed by Haber and Stornetta is referred to as *linear* linking scheme [11]. In order to diminish the need for trust, the time stamping authority links all time stamps together into a chain using the collision-resistant hash function h . In this case the time stamp for the n th submitted document y_n is $s = sig_{TSS}(n, t_n, ID_n, y_n, L_n)$, where t_n is the current time stated by the time

stamping authority, ID_n is the identifier of the submitter and L_n is the linking information defined by the recursive equation

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, h(L_{n-1})).$$

There are several complications with the practical implementation of this scheme. The major one is that the number of steps needed to verify the one-way relationship between two time stamps is linear with respect to the number of time stamps between them. Hence, a single verification may be as costly as it was to create the whole chain. It has been shown that this solution has impractical trust and broadcast requirements.

Haber and Stornetta proposed a *tree-like* scheme [12], based on Merkle's authentication trees [16,17], in which the time stamping procedure is divided into rounds. Since the tree-like scheme is feasible, we used this solution to implement our time stamping system.

4 System Architecture and Implementation

We designed and implemented an architecture for providing time stamping services with emphasis on interoperability, since our main goal was to enable inter-enterprise integration of time stamping services.

We adopted a multi-tier architecture, since this facilitates a development approach exploiting clean separation of responsibilities, and makes the proposed solution more flexible. The middle-tier components and the back-end servers are located in the first and second tier of the architecture, respectively.

The rest of this section is organized as follows. Subsection 4.1 thoroughly explains the back-end structure and the mechanisms by which the system provides relative temporal authentication and absolute temporal authentication. Such mechanisms are based on a linear linking scheme, and on publishing mechanisms as well as on a trusted time source, respectively. Subsection 4.2 provides implementation details on the middle-tier, including the specific technologies we have used for the development and for the deployment of the system prototype.

4.1 The Back-End

The back-end satisfies the application functional requirements, in that it implements cryptographic routines for the provision of time stamping functions. The overall structure of the back-end server is depicted in Figure 2.

The basic functions the back-end time stamping system provides are: creation of a time stamp, verification of existing time stamps, publication of global hash values for auditing and independent verification.

In the following, we briefly describe the roles of individual back-end components.

The *Proxy component* is in charge of handling communication with the back-end.

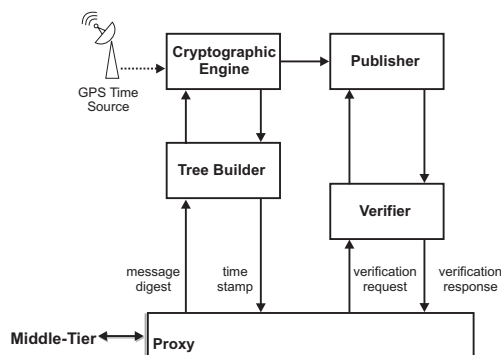


Fig. 2. The organization of the back-end.

The *Tree Builder* component accepts a sequence of input time stamping requests and builds a single time stamping request which is submitted to the *Cryptographic Engine*. The *Cryptographic Engine* adds the time value and executes the linking process. Building tree structures results in a performance improvement for the system, since the load which is delivered to the *Cryptographic Engine* is reduced to a fraction of the one which the *Cryptographic Engine* would experience if requests were sent directly to it. The *Tree Builder* component basically uses the Haber and Stornetta's tree-like scheme based on Merkle's authentication trees [16,17]. Each tree corresponds to a round of the time stamping process and includes time stamping requests falling in a time window of about ten seconds. Intra-round relative temporal authentication is not provided. Basically, Merkle's authentication tree is a method of providing short proofs (logarithmic in the number of inputs) that a bitstring x belongs to the set of bitstrings $\{x_1, x_2, \dots, x_n\}$. During a round a binary tree is constructed as follows (see Figure 3): the leaves are labeled with message digests extracted from the time stamping requests obtained during the round, every non-leaf is labeled with a message digest computed using a hash function h over a concatenation of the labels of its children.

Hence, the label of the root of the tree depends on all the leaves, i.e. aggregated bitstrings. For every leaf it is possible to prove this dependence by exhibiting some more vertices of the tree; the minimal collection of such extra nodes is called an *authentication path*. Figure 3 shows also an example of

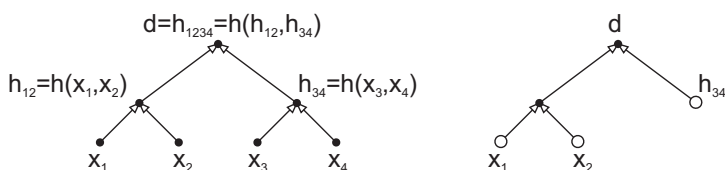


Fig. 3. Merkle's authentication tree and the authentication path for bitstring x_1 .

a Merkle's authentication tree where in order to prove that the root value d is dependent on the input x_1 it is enough to add vertices x_2 and h_{34} and compute $h_{12} = h(x_1, x_2)$ and $d = h_{1234} = h(h_{12}, x_{34})$.

The adopted hash function is obtained as a concatenation of two well-known hash functions, i.e. SHA1 [18] and RIPEMD160 [10]:

$$h(x) = \text{SHA1}(x) \& \text{RIPEMD160}(x)$$

We chose a compound hash function in order to strengthen the security of our time stamping server: in the event that one of the two hash functions is broken, the system will not be compromised and old time stamps will remain valid as long as the other hash function is secure.

The *Cryptographic Engine* component performs two distinct actions:

1. It freezes the time stamping request sequence by linking the consecutive rounds based on a linear scheme. Direct dependencies are created by computing message digests using hash functions.
2. It adds the time value to the time stamp to be formed. Such a value is provided by a trusted time source (such as a GPS clock).

Time stamps are signed by the server to assure their authenticity and instant verifiability.

The *Publisher* component computes a digest for the entire time window of related time-stamping requests, and makes it publicly available by some auditable media. In the current implementation such digests are computed and published daily.

Please note that the two actions performed by the Cryptographic Engine component and by the Publisher provide three different forms of time authentication: the linking process (action 1 of the Cryptographic Engine component) provides the fine-grain relative temporal authentication between time stamps; attaching the absolute time value to the stamps (action 2 of the Cryptographic Engine component) allows users to gain fine-grain information on the time elapsed between two published root values; the publication on a publicly available auditable media (action performed by the Publisher) provides coarse-grain absolute temporal authentication; publishing (action performed by the Publisher) is also one of the central means of retaining the long-term proof value of the time stamps, regardless of the trustworthiness of the time stamping service provider.

Note also that the absolute time is to be thought of as an additional piece of information provided by the system with respect to the basic linking scheme. Obviously, this additional feature relies on the security of the system private key. However, erroneous or fraudulent behaviors would be immediately detected and proved by means of the linking process. The *Verifier* component allows any external user to check the validity of old time stamps. An instant verification on a time stamp is always possible without interacting with the time stamping server by simply checking the digital signature which was provided with the stamp. However, as already mentioned, this method requires an unconditionally trusted time stamping server and critically relies on the security of the provider's

private key. The Verifier component instead provides the verifier with all the data needed to reconstruct a hash chain from the time stamp in question to the closest published hash value, so that the time stamping provider is not required to be trusted and the verification process is as trustworthy as the publishing media are.

As far as the implementation of the system prototype is concerned, the back-end code runs on a IBM Server X-370 with four Pentium Xeon 900 MHz processors, running a Linux kernel 2.4.12. The code is written in C++ language and is based on Crypto++ [24] library for cryptographic functions and on Snacc [25] based efficient C++ routines and data structures to support BER encoding and decoding of ASN.1 data structures.

4.2 The Middle-Tier

The middle-tier is in charge of leveraging services provided by the back-end server, in order to satisfy interoperability requirements which arise from the heterogeneity of the clients. It also decouples the implementation of the service (i.e., the back-end) from its interface. To this aim, the middle-tier wraps the services provided by the back-end. It forwards client requests to the back-end server, and delivers the servers replies to the clients.

Interactions between the middle-tier and the back-end take place via the Gateway (at the middle-tier side) and the Proxy (at the back-end side).

The middle-tier is also responsible for anonymity of requestors. Actually, an important property of a time stamping service is that it should never bind clients to time stamp requests.

The middle-tier was implemented using the the emerging Web Services technology to facilitate service access from any platform. The structure of the middle-tier is shown in Figure 4.

From the client prospective, the system appears as a Web Service provider. The service requestor can access the middle-tier via any transport protocol available over the Internet, provided that a protocol listener for the specific protocol has been implemented (in particular, the current implementation provides an HTTP listener, and FTP listener, which are two of the transports recommended in RFC3161 [6]). Requests and responses are exchanged through SOAP messages. The entry point for requestors' messages consists thus of a SOAP engine (see Figure 4) which coordinates the SOAP message flow through the subsequent components. The engine is also responsible for ensuring that the SOAP

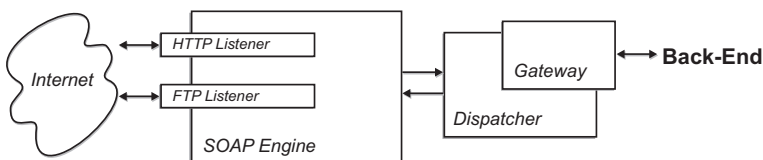


Fig. 4. The organization of the middle-tier.

semantics are followed. Clients are not aware of the implementation details of the service provided by the back-end components. All they know is which services are available and what their interface is like.

The *Dispatcher* component is responsible for acting as a bridge between the SOAP processor and the functional components. It identifies the Java objects to which to delegate the execution of individual activities, and invokes the appropriate methods on such objects.

The *Gateway* component is in charge of data conversion from XML and SOAP data structures to ASN.1 data structure (and viceversa). ASN.1 data structures are compliant with indications contained in RFC3161. An abstract of the time stamping request and of the time stamping token is reported in Fig. 5.

```

TimeStampReq ::= SEQUENCE {
    version                INTEGER { v1(1) },
    messageImprint         MessageImprint,
    reqPolicy              TSAPolicyId          OPTIONAL,
    nonce                  INTEGER              OPTIONAL,
    certReq                BOOLEAN              DEFAULT FALSE,
    extensions              [0] IMPLICIT Extensions OPTIONAL }

TimeStampResp ::= SEQUENCE {
    status                 PKIStatusInfo,
    timeStampToken         TimeStampToken      OPTIONAL }

```

Fig. 5. Abstract of data structures for the time stamping request and response.

The time-stamp token is the data item which contains the certified time information.

All layers of the presented multi-tier architecture work in a pipelined fashion to achieve high throughput.

The prototype version of the system we have built, is deployed within an IBM WebSphere Application Server version 4, on top of a Dell PowerEdge 1400SC with two 1400MHz Pentium III processors, running a Linux Red Hat kernel 2.4.18-3 with dual processor support.

5 Integration Experiment

To test the effectiveness of the proposed solution, we integrated a generic business application with the time stamping server we have implemented. The prototype application handles purchase order documents, whose submission time is to be certified.

Figure 6 shows the class diagram of the front-end code of the case study application.

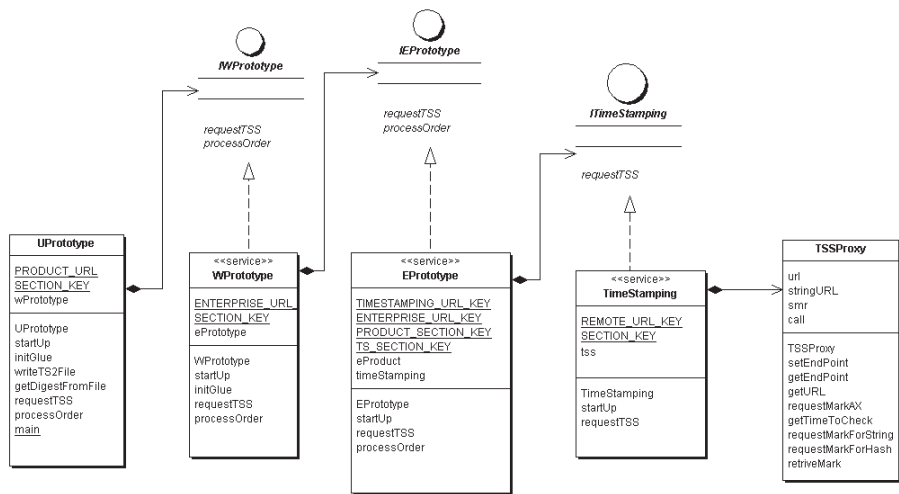


Fig. 6. Main classes for the experimental set-up.

The **UPrototype** class includes the utilities which performs the hash functions. Its input parameter is the path of the file to be processed, and its output parameter is the path of the file where the time-stamping token is to be stored.

The **WPrototype** class acts as a wrapper of the inner layers.

The **EPrototype** class contains the business logic for processing the purchase order data.

The **processOrder** method applies the specific proprietary business logic.

The **requestTSS** method calls the remote time stamping service.

The **TimeStamping** class represents the local component which requests and binds the Web service providing time stamping functionalities.

Figure 7 shows the sequence diagram for the request and delivery of a time stamp.

TSSProxy component acts as a local entry point to the remote service provider. The requestor submits her or his document to the system and passes the return path for the time stamp token. The system takes care of performing the hash functions and calling the remote web service. Once the token is received, the requestor can add the certified time to the order and go on with the remaining order processing operations.

We explicitly note that the interaction between the **TSSProxy** and the remote time stamping server is asynchronous. The parameter which regulates this interaction is the time-to-check-back, i.e. the time that the **TSSProxy** has to wait between the moment when the request is forwarded to the time stamping service, and the moment when the response will be available.

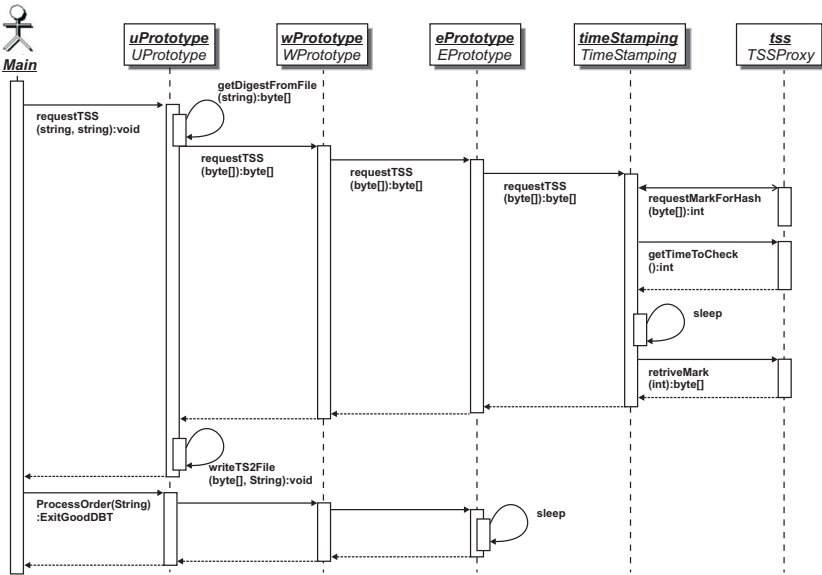


Fig. 7. Main success scenario for the request and delivery of a time stamp.

6 Conclusions

This paper described the results of a research activity conducted cooperatively by an academic and an industrial party. It presented a practical solution for and an experience in the implementation of time stamping services and their exposition to the Internet for inter-enterprise integration. The paper discussed state-of-the-art time stamping algorithms and crucial issues related to their practical implementation. Focus was on integration problems which arise when a potentially large community of enterprises – which rely on a handful of heterogeneous technologies – is willing to access remote third-party time stamping services.

We proposed a practical architecture for providing time stamping services across enterprise boundaries. The architecture relies on a multi-tier structure. The back-end tier provides the basic time stamping functionalities and is based on the Haber and Stornetta linking scheme. The time stamping process is divided into rounds based on tree-like structures. The system provides both relative temporal authentication, based on the linking scheme, and absolute temporal authentication, based on publishing mechanisms as well as on a trusted time source. The middle-tier is in charge of leveraging services provided by the back-end server, in order to satisfy interoperability requirements which arise from the heterogeneity of the clients. It has been implemented using the emerging Web Services technology. Messages are compliant to RFC3161. An integration experiment has been conducted to evaluate the effectiveness of the proposed solution. We integrated a generic business application with the time stamping server we have implemented. Experimental results have shown that the Web

Services technology is well-suited for delivering time stamping services to the users of an inter-enterprise infrastructure.

Acknowledgements. Authors are grateful to Annalisa Caso, Sonia Del Vacchio, Francesco Festa, and Gerardo Maiorano for the numerous fruitful technical discussions, and for developing the bulk of the code.

This work was partially funded by: the University of Naples Federico II, Sintel SPA, the Consorzio Interuniversitario Nazionale per l'Informatica (CINI), the National Research Council (CNR), and the Ministry of University and Research (MIUR), within the framework of projects: SP1 "Sicurezza dei documenti elettronici", "Oltre la Firma Digitale" (OFD), "Gestione in sicurezza dei flussi documentali associati ad applicazioni di commercio elettronico", and "Middleware for advanced services over large-scale, wired-wireless distributed systems" (WEB-MINDS).

References

1. Clarke, F., Ekeland, I.: Nonlinear oscillations and boundary-value problems for Hamiltonian systems. *Arch. Rat. Mech. Anal.* **78** (1982) 315–333
2. Clarke, F., Ekeland, I.: Solutions périodiques, du période donnée, des équations hamiltoniennes. *Note CRAS Paris* **287** (1978) 1013–1015
3. Michalek, R., Tarantello, G.: Subharmonic solutions with prescribed minimal period for nonautonomous Hamiltonian systems. *J. Diff. Eq.* **72** (1988) 28–55
4. Tarantello, G.: Subharmonic solutions for Hamiltonian systems via a $\mathbb{Z}_p$ pseudoindex theory. *Annali di Matematica Pura* (to appear)
5. Rabinowitz, P.: On subharmonic solutions of a Hamiltonian system. *Comm. Pure Appl. Math.* **33** (1980) 609–633
6. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP) Available at <http://www.ietf.org/rfc/rfc3161.txt> (2001)
7. Bayer, D., Haber, S., Stornetta, W., S.: Improving the efficiency and reliability of digital timestamping. *Proceedings Sequences II: Methods in Communication, Security, and Computer Science*. Springer-Verlag (1993) 329–334
8. Benaloh, J., de Mare, M.: Efficient Broadcast time stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science (1991)
9. Buldas, A., Laud, P., Lipmaa, H., Villemson, J.: Time Stamping with Binary Linking Schemes. *Advances in Cryptology CRYPTO '98* (1998)
10. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160, a strengthened version of RIPEMD, Fast Software Encryption. LNCS 1039, D. Gollmann, Ed., Springer-Verlag (1996) 71–82
11. Haber, S., Stornetta, W., S.: How to timestamp a digital document. *J. of Cryptology*, **3** (1991) 99–111
12. Haber, S., Stornetta, W., S.: Secure Names for Bit-Strings. *Proceedings of the 4th ACM Conference on Computer and Communications Security*. (1997) 28–35
13. Housley, R.: Cryptographic Message Syntax (CMS). Available at <http://www.ietf.org/rfc/rfc2630.txt> (1999)

14. International Organization for Standardization and International Electrotechnical Commission: ISO/IEC Standard 18014: Information technology - Security techniques -Time stamping services (2002)
15. Kreger, H.: Web Services Conceptual Architecture. IBM Software Group. Available at <http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf> (2001)
16. Merkle, R., C.: Protocols for public key cryptosystems. Proceedings of the 1980 IEEE symposium on security and privacy, IEEE Computer Society Press (1980) 122–134
17. Merkle, R., C.: A certified digital signature. Advances in cryptology - Crypto'89, Springer-Verlag **435** (1989) 218–238
18. National Institute of Standards and Technology: Secure Hash Standard, FIPS PUB 180-1. Federal Information Processing Standards Publication. Available on-line at <http://www.itl.nist.gov/fipspubs/fips180-1.htm> (1995)
19. Rivest, R.: The MD5 Message-Digest Algorithm, RFC 1321. MIT LCS & RSA Data Security Inc (1992)
20. Sneed, H., M.: Encapsulating legacy software for use in client/server systems. Proceedings of Working Conference on Reverse Engineering (1996) 104–119
21. Surety, Inc., Herndon, Virginia. www.surety.com
22. Zhao, W., Moser, L., E., Melliari-Smith, P., M.: Design and Implementation of a Pluggable Fault Tolerant CORBA Infrastructure. Proceedings of the International Parallel and Distributed Processing Symposium (2002) 343–352
23. Zhao, W., Moser, L., E., Melliari-Smith, P., M.: Design and implementation of a consistent time service for fault-tolerant distributed systems. Proceedings of the International Conference on Dependable Systems and Networks (2003) 341–350
24. Crypto++ Project: <http://www.eskimo.com/~weidai/cryptlib.html>
25. Snacc website: <http://www.fokus.gmd.de/ovma/freeware/snacc/entry.html>

A Lightweight Security Model for WBEM

Giuseppe Cattaneo, Luigi Catuogno, Umberto Ferraro Petrillo, and
Ivan Visconti

Dipartimento di Informatica ed Applicazioni
Università di Salerno
Via S. Allende
84081 Baronissi (SA) ITALY
`{cattaneo,luicat,umbfer,visconti}@dia.unisa.it`

Abstract. Web-Based Enterprise Management (WBEM) is an emerging standard solution for remote management of heterogeneous devices. It allows to remotely operate and administer a group of hardware and software devices while preserving some security features. The aim of this paper is two-fold: 1) We raise concerns regarding security weaknesses of the architecture of WBEM. 2) We propose a lightweight security model for WBEM based on the concept of Attribute Authority and show its effectiveness in preserving both the security and the performance of the system. Moreover, we address the concept of accountability and present guidelines for an implementation of our model.

1 Introduction

The exponential growth of services accessible by means of a telecommunication network is driving an increasing amount of investments and resources in the area of *Network Management*. We refer, by this term, to the set of tasks that are accomplished in order to administer a networked system and to collect and handle any sort of information that the components of the network could ever propagate. Historically, the majority of network management technologies has been designed to be deployed within single and trusted administrative domains. In many cases, the resources to be managed (both hardware and software) were developed using proprietary and extremely specialized technologies to cover a well defined application scenario.

Nowadays, several factors raise the need to disclose and to standardize network management technologies. Among them, we identify three main points. First: the increasing need of sharing management tasks among several actors. Second: the need for a remote seamless administration of network components developed by different vendors that have to coexist into the same domain. Third: the increasing use of segments of third-party networks in order to interconnect several components of the same private infrastructure.

Many international standardization boards together with several consortia of corporations, have worked on definitions of common sets of standards and design/implementation guidelines, in order to allow interoperability with products

developed by third parties and even by their direct competitors. This work has produced a wide set of technologies that involve each aspect of the network management. Among these we focus our attention on WBEM[3], that is an emerging solution for the network management problem.

The Web-Based Enterprise Management (WBEM) is a set of Internet and management standards designing a framework for the distributed administration of hardware and software devices. It designs a middleware that allows applications to administer several heterogeneous remote devices (accessible by means of an element manager software). Communications between client applications and element managers is handled by an intermediary software layer that we refer to as the *WBEM core*.

1.1 Motivations

We believe that the current WBEM architecture is not able to cope with a scenario, explained in the following, that is becoming quite common.

We are witnessing an increasing interest of the telecommunication world into the possibility of sharing or renting a communication device on a time-limited basis. Such considerations apply also to the management domain.

To this end, suppose there exist two institutions, respectively institution $\mathcal{A}$ and $\mathcal{B}$ where $\mathcal{B}$ has to be allowed to access a device belonging to $\mathcal{A}$. It is desirable for $\mathcal{B}$ not only to access such a device but even to monitor its status as well as to adjust its configuration. The problem that arises is to decide which is the optimal way for $\mathcal{A}$ to grant management access to shared devices to institution $\mathcal{B}$ while enforcing access policies (see Figure 1). Another issue for $\mathcal{A}$ concerns with the performance of its own WBEM core. As it could be easily guessed, the WBEM core is the bottleneck of the whole management architecture as every management operation issued by any client must be processed by this component before being committed. This poses a serious scalability issue when the number of client applications and supported managed devices increases. Starting with these premises we have to drop the simplest solution, i.e. letting users of $\mathcal{B}$ to directly access to the WBEM core belonging to $\mathcal{A}$. On the one side this solution has the advantage of not requiring any change on the current WBEM architecture. On the other side, it does not allow $\mathcal{B}$ to enforce any fine-grained access control policy over its employers while it pushes more work on the WBEM core of $\mathcal{A}$. A better alternative would be to let users of $\mathcal{B}$ interact with the local WBEM core. Then, the WBEM core of $\mathcal{B}$ would interact with the WBEM core of $\mathcal{A}$ by qualifying itself as a client application. In such a way it would be possible either for $\mathcal{B}$ to enforce a local policy on its employers either $\mathcal{A}$ would be still able to control accesses coming from $\mathcal{B}$. The drawback of this solution is that it requires the WBEM core of $\mathcal{A}$ to process all requests coming from the other institutions. In order to preserve efficiency, the unnecessary and dangerous hop on the $\mathcal{A}$'s WBEM core should be avoided. Hence, $\mathcal{B}$'s WBEM core should be allowed to directly manage devices belonging to $\mathcal{A}$. In this last scenario, several security issues arise.

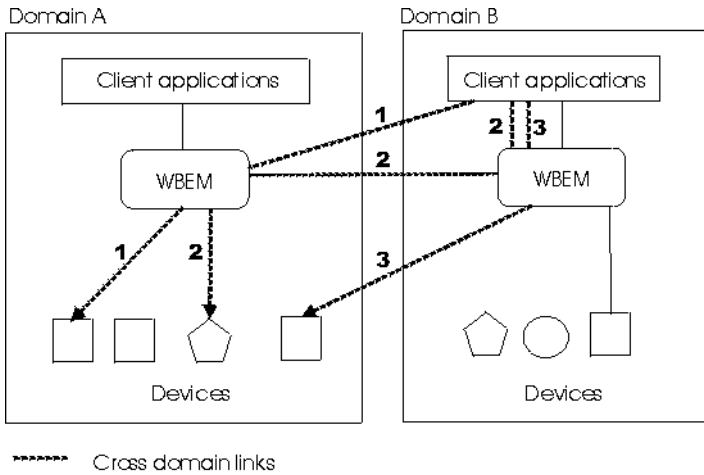


Fig. 1. $\mathcal{A}$ rents to $\mathcal{B}$ some resources: three possible alternatives. The first alternative represents the trivial solution, that is, client applications in the domain $\mathcal{B}$ interact with the rented devices using the WBEM core of $\mathcal{A}$. Access policies are enforced only by the WBEM core of $\mathcal{A}$. In the second alternative, client applications interact only with their local WBEM core that in turns accesses the shared resources through the WBEM core of $\mathcal{A}$. Access policies are enforced by the WBEM cores of $\mathcal{A}$ and $\mathcal{B}$. In the third alternative, the WBEM core of $\mathcal{B}$ is allowed to directly access the shared resources. In this case, access policies are enforced by the WBEM core of $\mathcal{B}$ and by the management software providing access to resources exported by $\mathcal{A}$.

Security Issues. In the example described above the organization $\mathcal{A}$ gives to the organization $\mathcal{B}$ the permission to access some of its devices. Such a kind of transaction typically occurs when an organization (in our case $\mathcal{A}$) wants to rent a resource that is temporarily unused.

The main issues in these *cross-domain* interactions, is that both $\mathcal{A}$ and $\mathcal{B}$ have to trust each other. In other words, $\mathcal{A}$ should be guaranteed that $\mathcal{B}$ will be not allowed either to perform operations beyond his permissions, or access $\mathcal{A}$'s private data, but on the other hand, $\mathcal{B}$ has to be guaranteed that his legitimate requests to the equipments of $\mathcal{A}$ should be correctly performed and moreover, correctly accounted.

We focus on some security issues that are not considered in current proposals and thus menace their dependability. In our example, $\mathcal{A}$ should be protected against malicious activities performed by $\mathcal{B}$. For example such activities could be performed in order to decrease the performance of the WBEM core of $\mathcal{A}$ as a Denial of Service (DoS in short) attack.

This discussion strongly motivates the necessity that the WBEM core of $\mathcal{A}$ have not to deal with operations performed by $\mathcal{B}$'s applications and WBEM core. Furthermore, in order to correctly bill $\mathcal{B}$'s activities, $\mathcal{A}$ wishes to be guaranteed that each request issued by $\mathcal{B}$ is *undeniable*. On the other hand, $\mathcal{A}$ could be dishonest in executing, accounting and billing $\mathcal{B}$'s requests and even results

notification could be altered. In this case, $\mathcal{B}$ wishes to have the proof that each action it sends to $\mathcal{A}$ will be correctly processed, that any $\mathcal{A}$'s response will be also undeniable and that $\mathcal{A}$ cannot generate fake requests of behalf of $\mathcal{B}$.

1.2 Our Result

In this paper we present a lightweight security model for the WBEM middleware for securing remote cross-domain administration. More precisely, we propose an extension of the WBEM internal communication protocol that efficiently guarantees security properties as unforgeability and undeniability. Our security model is based on *attribute certificates* to implement access control without decreasing the performance of the WBEM core.

2 Background

In this section we briefly introduce the technologies that are involved in our security model.

2.1 WBEM

WBEM is a set of management and standard Internet technologies developed to unify the management of enterprise computing environments. It makes it possible to write management applications able to transparently administer heterogeneous devices. The core of WBEM is the Common Information Model (CIM)[4]. It is an object oriented model defining a technology-independent description of the devices to be managed and of the management operations they should support. By adopting CIM, it becomes possible to draw a borderline between supporting management operations and writing management applications. The former can be achieved by implementing the objects and the operations as described in the CIM model. The latter can be achieved by instantiating and using objects as described in the same CIM model.

The WBEM architecture adopts a distributed approach where several agents (here referred to as providers) managing software or hardware devices wait for instructions from the CIM Object Manager running as the core of the WBEM middleware. This component maintains a CIM-compliant representation of the managed devices. Management applications are able to administer a device by requesting to the WBEM core the execution of methods on the corresponding CIM objects. Such requests are forwarded by the WBEM core to the proper provider. After committing these operations, the corresponding object provider will notify their return values to the WBEM cores. The WBEM core will then notify the return value of each operation to the originating client. Communication among different parties in the WBEM architecture is accomplished using the HTTP protocol. The providers may be coded as in-process activities of the WBEM core or as external autonomous processes, possibly running on different machines with respect to the WBEM core. In order to control the access to

managed resources, the WBEM architecture requires a preliminary authentication step where each user provides his credentials. Moreover, support for secure communications is guaranteed by means of encrypted socket connections.

Security Model. The original WBEM framework did not address many security issues. The most part of the available security features were simply inherited for free by the adopted technologies (e.g., secure communication available through the TLS [6] protocol). One of the most critical issues of WBEM is the lack of any access control service. Access control is needed for selectively allowing accesses to managed devices. In order to solve this issue, the DMTF formed the User Security Working Group (USWG). The goal of this group is to provide a set of relationships between the various representations of users, privileges and managed resources. The USWG released a preliminary white paper (see [2] for details) detailing a *Security Model* to be added to the WBEM architecture. This model is mainly covers management functions such as administering users' accounts and maintaining and enforcing policies for controlling access to managed resources. The current specification of the *Security Model* defines only a limited set of features. The most significant one is the access control policy. It allows one to define the authorization policies with respect to the identity of the user issuing a request, to the resource specified by the request and to the type of operation to be performed. From the architectural point of view the system uses an LDAP [18] service in order to store and to gather policies and user accounts. Clients' requests for management operations are validated by the WBEM core according to the corresponding policy. Successfully validated requests are then committed to the qualified provider.

2.2 PKIX and Attribute Certificates

PKIX [8] is a public key infrastructure based on X509 digital certificates and is currently the standard PKI on the Internet (e.g., the TLS protocol [6] uses digital X509 certificates). An X509 digital certificate contains information about an entity along with his public key, a serial number, a validity period, a reference to a certification authority that has released the certificate and a digital signature computed by the certification authority.

X509 digital certificates can be used for user identification since only the owner of the certificate can give a proof of knowledge of the private key corresponding to the public key encoded in the certificate. The use of digital certificates for activities as signing contracts and voting strongly discourage the *lending* of the private key to other users.

Version 3 of X509 digital certificates supports the encoding of privileges, organizational roles and privileges in the certificate. Indeed a new section called *Extension* of the certificate can contain new attributes that can be used for application-specific purposes. However such an use of new attributes has two main drawbacks:

- **privacy:** even if the certificate is used for different purposes all attributes are anyway exhibited;

- **expiration:** the validity period of all encoded attributes corresponds to the one of the certificate.

In order to avoid such drawbacks the concept of *attribute certificate* has been proposed as an integration of PKIX. In this proposal each user has an X509 digital certificate and some attribute certificates. Each attribute certificate contains a unique reference to the X509 digital certificate (this can be easily achieved by encoding a reference to the certification authority and the serial number of the certificate) a validity period and one or more attributes. An attribute certificate is released by an *attribute authority*, i.e. an organization that verifies that a user possesses some privileges and an X509 digital certificate. The attribute authority releases an attribute certificate in which such privileges and the reference to the X509 digital certificate are encoded. The binding of the attribute certificate with the X509 digital certificate is necessary in order to discourage the lending of privileges. The attribute certificate is signed by the attribute authority but does not contain any user public key. Indeed when a user wants to prove possession of privileges he sends both the X509 and the attribute certificates and gives a proof of knowledge of the private key corresponding to the public key encoded in the X509 digital certificate. The party that receives such a proof verifies that both certificates are well formed, signed by trusted authorities and not yet expired and then verifies the proof of ownership given by the user.

Attribute certificates solve both drawbacks described above regarding the use of the *Extensions* section of X509 v.3 digital certificates for privilege management. Indeed each time that possession of a privilege has to be proved only the attribute certificate that corresponds to such a privilege must be exhibited and its validity can be shorter than the one of the X509 digital certificate.

3 Related Work

The idea of providing selective access to management functionalities by means of some authorization or authentication credentials is not new. Several contributions exist in this area leading to the definition of “*Trust Management*”. Blaze *et al.* [1] define the “*Trust Management*” as a set of activities and criteria that should be performed by any architecture for the distributed management of shared resources. More in detail, Trust Management activities can be summarized in definition of security policies, definition and description of privileges, verification of compatibility between actions required and privileges provided and criteria for trust agreement with third parties. Khurana and Gligor [5] strongly motivate the deployment of attribute certificates in trust management frameworks, and specifically the need to implement the attribute authority inside the management domain (without any third party involved).

Akenti[20] is a system for enforcing certificate-based access policies in a PKI environment. Authorization policies for a resource are represented as a set of digitally signed certificates. These certificates are created by stakeholders and

specify which attributes a user must have in order to access a resource and who is trusted to make such use-condition statements. Whenever an access request is issued, the Akenti *police engine* gathers all relevant certificates from both the user and the resource, validates them, and determines the users rights with respect to the resource. The Akenti system designs two models for enforcing access policies. In the *authorization model* the access to shared resources is controlled by a *resource gateway*. Client applications issue access requests toward the *resource gateway* by means of TLS connections providing their X.509 certificates. Then, the *resource gateway* queries the *police engine* to know if the user is allowed to perform the requested operation. In the *capability model* client applications are allowed to directly query the *police engine* providing the identity of the target resource they want to access. As a result, the *police engine* will return a signed capability certificate stating which are the permissions granted to that client for accessing the requested device. The client application exhibits the capability certificate to the *resource gateway* in order to gain access to the target resource. In order to avoid the problems of revocation the capability certificates are short-lived.

SPKI [12] is a standard for digital certificates that focuses on authorization rather than authentication to make trust management simple and extensible. Using SPKI a certificate carries some privileges and has not to be linked to another certificate (as it happens with attribute certificates).

Delegation certificates [10] designed on top of Impersonation certificates [11] have been proposed in order to implement a system for propagation of authorization information. In this model the rights of a user U are encoded in a digital certificate along with his key. U can release other certificates and in each of them a subset of his rights can be encoded. Obviously the concept of attribute authority implemented inside an organization that possesses some resources and gives rights to access them to other organizations by means of attribute certificates, strictly overlap with the concept of delegation certificates. The main difference that suggests/discourages the use of one of the two proposals is the advantage/weakness of giving to a party some rights that it can even partially lend to other parties. For an overview on the use of delegation certificate for distributed access-rights management see [9].

In [13] a distributed security infrastructure that supports access control based on authorization roles is presented. Their work organizes some of the proposals presented in [12,1] and specifies an high-level distributed system with several useful properties regarding access control lists and security policies.

In this work we discuss some security issues of the current WBEM architecture and we will use the simple concept of attribute certificate to design a lightweight security model that addresses such weaknesses. The advantage of our model with respect to past solutions is its smaller impact on the WBEM model both in terms of architectural and computational complexity.

4 Our Proposal

In this section we describe our security model for management of remote devices based on WBEM. In our scenario an organization $\mathcal{A}$ wants to give to an organization $\mathcal{B}$ (however our model can be immediately extended to a context in which several organizations co-exist) a temporal access to some resources managed by a provider $\mathcal{D}$. Our architecture enjoys the following properties:

1. **Unforgeability:** only the resources of $\mathcal{A}$ for which access has been granted are given by $\mathcal{D}$ to $\mathcal{B}$, even in case $\mathcal{B}$ is corrupted and arbitrarily deviates from the protocol. Moreover the lending of such privileges should be inconvenient for $\mathcal{B}$.
2. **Accountability:** each request performed by $\mathcal{B}$ and satisfied by $\mathcal{D}$ is undeniable. Moreover $\mathcal{B}$ cannot make a fake reply on behalf of $\mathcal{D}$ and a coalition composed by $\mathcal{A}$ and $\mathcal{D}$ cannot make a fake request on behalf of $\mathcal{B}$.
3. **Performance:** the performance of the WBEM core of $\mathcal{A}$ are not affected by the amount of work that is needed to satisfy the requests of $\mathcal{B}$.

The first property states that our model does not allow a cheater to obtain an unauthorized access to a resource of an organization $\mathcal{A}$.

The second property states that in case of dispute, $\mathcal{B}$ has a *witness* for proving that $\mathcal{D}$ has satisfied a request while $\mathcal{D}$ has a *witness* for proving that $\mathcal{B}$ has requested a resource. Moreover each authorized application that uses the infrastructure of $\mathcal{B}$ in order to access to a restricted resource can detect fake replies generated by a malicious activity of $\mathcal{B}$. The enjoying of the accountability property allows to compute the cost of accessing the resources of $\mathcal{D}$ considering the real use of the service and not only on its availability.

The last property states that the overhead inferred by such transactions must not affect the WBEM core of $\mathcal{A}$.

We describe three steps that formalize the execution of such a kind of transaction in our architecture. The first step, referred to as “Set-up”, is performed when $\mathcal{A}$ decides to give to $\mathcal{B}$ a temporal access to some resources managed by $\mathcal{D}$. The second step, referred to as “Access to the system”, concerns with the activity performed by a client application $\mathcal{B}$. The last step, referred to as “Access to a resource”, concerns with the activities performed by $\mathcal{B}$ and the provider $\mathcal{D}$.

The messages exchanged during a typical transaction are depicted in Figure 2. In our notation P is a generic party, $\{M\}_P$ is a message M signed by P , X_P is the X509 digital certificate of P , A_P is an attribute certificate of P , C_P is the WBEM core of P .

In general many security properties can be guaranteed on a public channel using the TLS [6] protocol. However TLS works only on top of TCP and the communication between a WBEM core and a provider is not necessarily (even though most of the times it is) performed using TCP. Moreover the simple use of TLS does not help in guaranteeing some security properties as accountability and TLS does not directly deal with attribute certificates (the support of attribute certificates inside TLS has been proposed in [19], however currently such

a support has not still been integrated in TLS). Thus we have decided to embed our security properties directly in the messages that applications exchange. Moreover in case of TCP connections, the secure tunneling with TLS can be obviously performed in order to support data encryption and authentication.

4.1 Set-Up

This is the first step executed when a transaction between $\mathcal{A}$ and $\mathcal{B}$ starts. $\mathcal{A}$ rents some resources managed by $\mathcal{D}$ to $\mathcal{B}$ for a period of time T . We assume that T is the expiration of such an access. We also assume that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$ have digital certificates released by a trusted certification authority (the certification authority could be managed by $\mathcal{A}$).

$\mathcal{A}$ works as an attribute authority and gives to $\mathcal{B}$ the required privileges by means of an attribute certificate.

The attribute certificate is composed by the following fields:

1. the expiration T ;
2. the attributes that encode the privileges on the resources given by $\mathcal{A}$ to $\mathcal{B}$ for this transaction;
3. a unique reference to the X509 certificate of $\mathcal{B}$;
4. the digital signature computed with the private key of $\mathcal{A}$.

$\mathcal{B}$ stores the attribute certificate in its WBEM core and the authorized applications can use the resources rented by $\mathcal{A}$ using the WBEM core of $\mathcal{B}$.

The provider $\mathcal{D}$ has to be configured in such a way that only the X509 digital certificates released by the trusted certification authority and the attribute certificates released by $\mathcal{A}$ are considered valid, this can be easily achieved storing the certificate of the certification authority and the one of $\mathcal{A}$ in $\mathcal{D}$. Moreover this step is independent of the transaction and thus can be performed only once for all transactions.

4.2 Access to the System

This step concerns with the interaction between a client application and the WBEM core of an organization. This is the step in which our proposal is less invasive. Indeed, such an interaction is performed inside an organization (in our example $\mathcal{B}$) and thus each organization could freely implement it considering its peculiarities. We suggest the use of the TLS [6] protocol that, in this step, guarantees data integrity, authentication and identification of parties. Moreover the WBEM core of $\mathcal{B}$ could use a local database in order to grant or deny the requests of the application on the base of its privileges, this can be efficiently implemented by using an LDAP [18] service.

4.3 Access to a Resource

For each external organization $\mathcal{B}_i$ the provider $\mathcal{D}$ maintains a counter c_i that will be used against *reply* attacks. Such a counter can be initialized the first

time that $\mathcal{B}_i$ interacts with $\mathcal{D}$ since it could be expensive to update $\mathcal{D}$'s configuration during the *Set-up*. Thus the first time that an organization interacts with the provider the counter is set to 0. The provider has also a private key corresponding to the public key encoded in its X509 certificate and that will be used in order to sign messages. When the WBEM core of an organization receives from an application a request to be forwarded to a provider then, after the inside-organization verification process, it performs the following steps:

1. it prepends a counter c that is greater with respect to all previous used counters to the message;
2. it appends a digital signature to the request;
3. it appends to the request and to the digital signature the X509 and attribute certificates received by $\mathcal{A}$ that hold the privileges needed by the request.

When the provider receives a request from a WBEM core $\mathcal{B}_i$, it performs the following steps:

1. it verifies the digital X509 and attribute certificates sent by $\mathcal{B}_i$ by checking that they have been released and correctly signed by the trusted certification authority and by $\mathcal{A}$, that they are not expired and that each attribute certificate refers to the X509 digital certificate;
2. it verifies the digital signature appended by $\mathcal{B}_i$ to the request;
3. it verifies that the counter c in the request is greater than c_i ;
4. it checks the resources requested by $\mathcal{B}_i$ and verifies that the encoded privileges from the received attribute certificates are enough to access to the requested resources.

If each of the previous discussed steps succeeds then the request is satisfied by the provider and a message is sent back to $\mathcal{B}_i$. This message is composed by concatenating the request to the reply and appending a digital signature computed using the private key of $\mathcal{D}$.

5 Analysis

The purpose of our model is to satisfy the security issues described in Section 1.1 and, at the same time, to preserve the efficiency of the system. We show now that our system enjoys the unforgeability, accountability and efficiency properties.

Unforgeability. In our system the provider satisfies a request only if valid attribute certificates, in which the required privileges are encoded, are received along with a proof of possession of the corresponding private key. Thus using the implementations currently believed “secure” of the involved cryptographic primitives (e.g., RSA digital signatures) the providers of $\mathcal{A}$ cannot be cheated and the access control policy cannot be violated. Moreover, since $\mathcal{B}$ must use its private key, it is strongly inconvenient the lending of his privileges to external organizations since the private key allows complete control of the certificate even for malicious activities that damage $\mathcal{B}$.

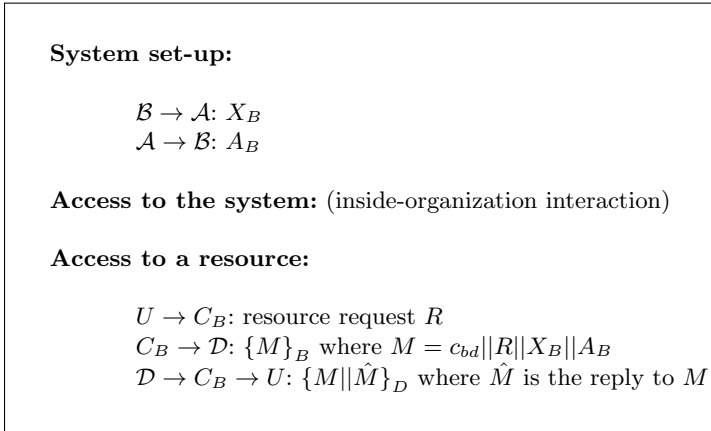


Fig. 2. The protocols executed by parties.

Accountability. Since each party appends a digital signature to each message to be sent, the standard techniques for preventing reply attacks (e.g., using a counter for each organization) and cut and paste attacks (e.g., using a session identifier to link all messages of a transaction) guarantee that the messages are undeniable and fake messages cannot be computed without the knowledge of the required private key. Thus the accountability property holds.

Efficiency. In our system the WBEM core of $\mathcal{A}$ is not involved in any of the three phases we have discussed above. Each time the resources are used the only involved parties are: the provider $\mathcal{D}$, that manages the resources and the WBEM core of $\mathcal{B}$, that uses them, while $\mathcal{A}$ is not involved.

6 Implementation Guidelines

In this section we show the viability of our proposal by describing the standard technologies useful to efficiently implement our security model. Next, we discuss some implementation hints that can be used to this end.

WBEM Implementations. Several implementations exist of the WBEM middleware. To just mention fews, we cite Windows Management Instructions (in short, WMI) from Microsoft [17], WBEM Services from Sun [16] and Pegasus from the Open Group [15]. In this paper we will focus on WBEM Services as it is currently the most mature open-source WBEM implementation available. The only concern with this implementation is that it does not actually allow a WBEM core to interoperate with a provider running on a different machine. Such a limitation is not technological and mainly comes from the choice of proposing WBEM as a local operating-system administration tool. It can be easily solved by allowing

the WBEM core to handle communication sockets with remote machines as well. Such a feature has been planned but not yet implemented in Pegasus while it is fully supported in the WMI implementation. WBEM Services closely follows the WBEM standard architecture except for the communication protocol. Namely, the RMI communication protocol is used in place of the standard HTTP based protocol in order to achieve a better performance and a better integration with previously existing Java-based components. The WBEM Services implementation available for the Solaris operating system adopts also a proprietary security model hinging on standard Solaris operating system services. Access control is enforced by using an approach that strongly resembles the one depicted by the USWG group (see Section 2.1) and essentially based on the adoption of an external LDAP server for storing the CIM objects access policies. For more information on this topic we refer the interested reader to [14].

Certification and Attribute Authorities. The implementation of a certification and an attribute authority can be performed by using the OpenSSL [7] package. This software package provides the implementation of a simple certification authority. Such an implementation can easily become an attribute authority by simply extending the list of fields that need to be included in an attribute certificate (this can be mostly done by just modifying a configuration file) and adding some procedures to check the correctness requirements of an attribute certificate. In addition, it is possible to efficiently support scenarios where access control policies frequently change by using short-lived attribute certificates; each of them can be released at cost of only few modular exponentiations to be performed by the attribute authority.

6.1 Implementing Our Model

The first step to be accomplished while implementing our model is the definition of a proper encoding for the permissions to be stored into attribute certificates. To this end, each CIM object that needs to be exported is referred to by means of a unique identifier. Such an identifier can be established by concatenating the name of the organization, the name of the provider and the name of the Object. In addition, we associate to every CIM object a permissions grid specifying for each method it implements the granted access attribute (i.e., read, write, read/write or deny). Hence, a typical attribute certificate will contain a list of the CIM objects it grants to access to (identified through their identifiers) together with a detailed description of access permissions.

The following step in the implementation of our model is the extension of both the WBEM core and the WBEM provider. We recall to this end that our model does not concern with the communication protocol between a client application and the WBEM core. Suppose we have a WBEM core that has obtained attribute certificates for accessing some resources managed by a remote provider and exported as CIM objects. Every time the WBEM core requests to the provider the creation a CIM object or the execution of a method upon an existing CIM object, it provides as an argument of the request its attribute

certificates. These will be used by the provider to check that the requesting WBEM core has enough privileges to issue that request. Note that it is possible to keep the cost of this check low by using the RSA digital signature scheme with a small public parameter. However, such an approach involves a fair amount of network overhead as the same attribute certificates have to be repeatedly sent every time an operation is invoked on a remote provider. A more complex and efficient approach would be to extend the existing communication protocol among the WBEM core and its providers by introducing an initial negotiation phase. During this phase, the WBEM core's attribute certificates are transferred to the target provider. These certificates will be cached by the provider and then used to validate any subsequent access request coming from the same WBEM core. Any access request received after the expiration of its referring attribute certificate will be handled by returning to the WBEM core a proper error message thus requiring a new negotiation phase.

7 Conclusion

In this paper we discussed about WBEM, a framework for the remote management of devices. We have outlined some serious security issues that arise in a scenario that is becoming quickly very frequent. We proposed a lightweight security model for remote management of devices providing several security properties. Moreover we addressed the concept of accountability that should be always considered when different organizations are involved in a distributed architecture. We presented a practical setting in which attribute certificates can be used in order to perform access control without the need of expensive interactions with an ad hoc service. Finally, we analyzed our system and described some guidelines that can be used to implement our architecture using current standard technologies.

References

1. Blaze, M., Feigenbaum, J., Lacy, J: Decentralized Trust Management. In Proc. of the 17th Symposium on Security and Privacy, May 1996, Oakland, CA. IEEE Computer Society Press, 164–173
2. Desktop Management Task Force: User and Security Model White Paper. <http://www.dmtf.org/standards/documents/CIM/DSP0139.pdf>, June 2003
3. Desktop Management Task Force: Web-based Enterprise Management. <http://www.dmtf.org>
4. Desktop Management Task Force: CIM Schema version 2.5. <http://www.dmtf.org>
5. Khurana, H, Gligor, V. D., Linn, J: Reasoning about Joint Administration of Access Policies for Coalition Resources. In Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), July 2–5, 2002, Vienna, Austria. IEEE Computer Society Press, 429–440
6. Dierks, T, Allen, C.: The TLS Protocol, version 1.0. Network Working Group, IETF RFC 2246, 1999
7. OpenSSL group: The OpenSSL Package. <http://www.openssl.org>

8. Housley, R., Ford, W., Polk, W., Solo, D: Internet X509 Public Key Infrastructure: Certificate and CRL Profile. IETF RFC 3280, April, 2002
9. Aura, T: Distributed Access-Rights Management with Delegation Certificates. *Lecture Notes in Computer Science*, Vol. 1603, 1999. Springer-Verlag 211–236
10. Engert, D., Thompson, M., Tuecke, S: Internet X.509 Public Key Infrastructure: Restricted Delegation Certificate Profile. Internet Draft, July 2001
11. Engert, D., Thompson, M., Tuecke, S.: Internet X.509 Public Key Infrastructure: Impersonation Certificate Profile. Internet Draft, July 2001
12. Ellison, C.: SPKI Requirements. IETF RFC 2693
13. Rivest, R, Lampson, B.: A Simple Distributed Security Infrastructure, April 1996
14. Sun Microsystems, Inc.: Solaris WBEM Services Administrator's Guide.
<http://docs.sun.com/db/doc/806-6468>
15. Open Group: Open Pegasus. <http://www.openpegasus.org>
16. Sun Microsystems, Inc.: Solaris WBEM Services.
<http://www.sun.com/software/solaris/wbem>
17. Microsoft Corporation: Learning about Windows Management Instrumentation (WMI). <http://msdn.microsoft.com>, July 1999
18. Wahl, M., Howes, T., Kille, S.: Lightweight Directory Access Protocol (v3). IETF RFC 2251, December 1997
19. Farrell, S.: TLS extensions for AttributeCertificate based authorization. Internet draft, August 1998
20. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K.: Certificate-Based Access Control for Widely Distributed Resources. In *Proc. of the 8th USENIX Security Symposium (SECURITY-99)*, August 23–26, 1999. Usenix Association 215–228

Security and Trust in Agent-Oriented Middleware

Agostino Poggi¹, Michele Tomaiuolo¹, and Giosuè Vitaglione²

¹ Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma,
V.le delle Scienze 181/A, 43100 Parma, Italy
{poggi, tomamic}@ce.unipr.it
<http://aot.ce.unipr.it>

² Telecom Italia Lab, Centro Direzionale di Napoli, isola F6, Napoli, Italy
giosue.vitaglione@telecomitalia.it

Abstract. Spreading of technologies as peer-to-peer networks, web services, multi-agent systems, ubiquitous computing, shows a clear trend toward open and highly distributed systems, requiring wide and dynamic overlay networks. But mass adoption of these technologies, especially in contexts where sensible resources are handled, will come true only if they will be able to guarantee a proper level of security. This paper presents the implementation of a security layer for JADE, a standard-based and widely deployed framework to build multi-agent systems. The proposed model, founded on delegation certificates, allows the management of trust relations among autonomous software agents collaborating and competing in wide and open agent societies, but it is also used to protect the underlying distributed infrastructure, providing required resources and services to hosted agents. Our approach, where policies are based on keys instead of names, enhances robustness and scalability by greatly reducing the impact of certificate directory unavailability on the authorization mechanism.

1 Introduction

To become an appealing solution for building security-critical applications, agent middleware should provide mechanisms to ease the management of evolving trust relationships, allowing agents to interact with other agents without facing security breaches, especially when they form large societies with changing social and economical conditions. In this context, when building middleware to support large and open agent-based systems, security, robustness and scalability are major issues.

Agent-based applications often rely on delegation of goals and tasks among trusted parties. In fact complex behaviors can emerge only in large societies of agents, cooperating and competing to achieve their own goals. Security-critical applications usually suppose the existence of different parties that have different and probably contrasting objectives. Agents should not be able to pose threats to their competitors, but they should be able to effectively collaborate with their partners. This can be accom-

plished if each delegation of duties is accompanied by a corresponding delegation of permissions, required to complete the task or achieve the goal.

This paper presents a security model where delegation certificates are the foundation of a distributed security infrastructure where trusted third parties and centralized directories of names are avoided. Trust management principles are applied to agent-based systems to realize systems that can implement secure access control mechanisms. All trust relationships can be founded on solid local believes, without relying on global directories of names and globally trusted certification authorities. In fact both of them make the system more centralized and may introduce additional points of breach, especially if their politics are not known in detail.

In particular, section 2 describes a generic model of distributed multi-agent systems. It shortly lists threats that can be posed by internal or external entities to protected resources. Section 3 evaluates delegation certificates, local names and trust management principles as a feasible alternative to globally trusted name directories and access control lists. Finally, section 4 presents the implementation of the security infrastructure of a real system. General principles exposed in previous sections are applied to JADE, a framework to build distributed multi-agent systems, protecting the access to sensible resources both at the application level and at the infrastructure level.

2 Security Threats in a Distributed MAS

Abstracting from other details and highlighting the components that can take reciprocal malicious actions, a distributed multi-agent system can be modeled through two different components:

- **Agents:** in its very essence, an agent can be thought as an atomic component with an associated thread of execution; an agent can communicate with local or remote agents (through *ACL messages*, i.e. *Agent Communication Language messages*, defined by the FIPA standard) as well as with its hosting environment (container) by means of local method invocations; each agent also exposes methods to be managed by the underlying container and relies on it to send and receive messages and to access needed resources;
- **Containers:** they constitute the local environment where agents are hosted and provide them several services; one of their main duty is to provide agents with an *ACC (Agent Communication Channel)*, so they can exchange messages; to complete their various tasks, containers can communicate on public networks with other containers and different remote components; containers can also host *message transport protocols (MTPs)* to allow communications with agents living on remote hosts.

Even if the environment where agents are hosted is often referred to as a *platform*, we will try to avoid this term. In fact while a platform, as defined by FIPA specifications [10], can be in general constituted by different distributed components, we define a container as a local runtime environment to be handled as an atomic component

from the security point of view. In JADE [11] a platform is a collection of containers typically distributed across several hosts, hosting a certain number of agents.

As fighting against security threats concerns, the final goal should be to have all interfaces exposed by agents and containers (represented in Figure 1) masked, both at the infrastructure level (interactions among remote components involving network communications) and at the agent level (actions on hosting environment and ACL message exchanges), so that specific actions are performed only if adequate security conditions are met. Each method invocation as well as any request delivered by an ACL message should be considered a threat to be accurately analyzed before granting access. A detailed classification of treats and proposed countermeasures is provided in [9].

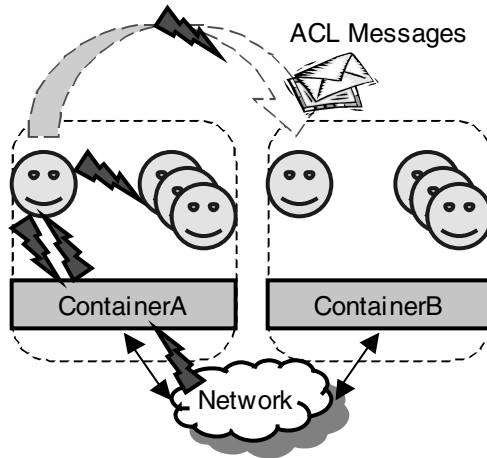


Fig. 1. Security threats in multi-agent systems

3 Access Control in a Distributed MAS

Traditional security frameworks take their decisions about authorizations to services and resources by using access control lists and identity certificates issued by globally trusted authorities. But weak and transient trust relationships, and corresponding delegations of permissions among trusted components, cannot be easily managed with access control lists (ACLs). Moreover, relying on trusted third parties introduces an additional point of security weakness for the whole systems. These concerns are becoming more and more important as relations among components providing on-line services can easily disappear, and new ones rise, as soon as social or economical conditions change. Peer-to-peer and ubiquitous computing trends may only exacerbate fickleness of relations among distributed components.

Moreover, appeal of agents comes from their ability to interact to achieve common goals. Agent-based applications often rely on delegation of tasks and goals among cooperating parties. These delegations of duties require corresponding delegations of permissions needed to perform required tasks or to achieve required goals. Delegated permissions could be used on their own, or joined with permissions obtained in other ways. While managing a local resource, an agent could choose to exploit access rights delegated by current requester, perhaps joined with a subset of its own access rights, but not to exploit permissions received by other requesters. Staging these behaviors in a system based on access control lists is not simple.

3.1 Delegation Certificates

Our approach to enforce platform security restricts access control lists, or policy files, to a small number of pre-defined entries, each one linking a trusted principal to a set of permissions. These permissions, quantifying a level of trust between platform security manager and known principals, are then packed in signed certificates and distributed to authenticated principals.

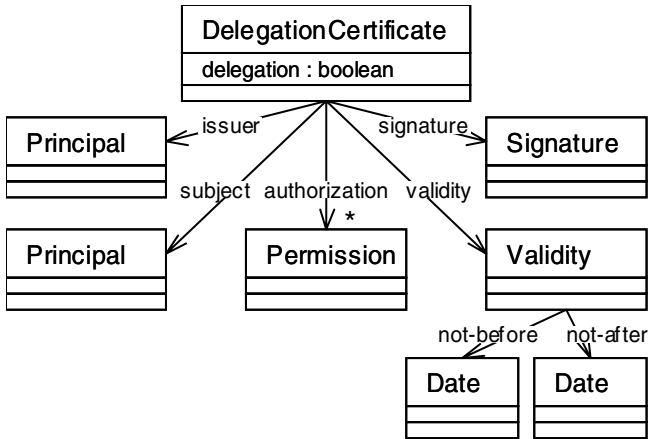


Fig. 2. Structure of delegation certificates

Essentially through a delegation certificate an issuer can grant a subject access to a resource (if allowed itself). As shown in Figure 2, each certificate carries a list of permissions, a delegation flag, and a validity period, to regulate the amount of trust that is transferred.

If a delegation certificate has its delegation flag set, then the subject of the certificate can further delegate received access rights to another subject. Even if this should happen on the basis of sound relationships, either technical or economical ones, each principal is free to choose its trusted partners.

The possibility to delegate permissions paves the way for a distributed management of access rights, which mimics security policies based on access control lists,

but as a result of a bottom-up process and without relying on a large centralized repository.

In this process, certificate chains take form, allowing access rights to flow from original issuers (resource managers) to final subjects (users of protected resources). Moreover, when different chains intertwine, certificates can dynamically form complex graphs, called *delegation networks*, as to fit evolving trust relations. A full theory of delegation networks is developed in [4].

3.2 Key Based Access Control

Certificates can be used to implement access control in different ways. One way (shown in Figure 3) is to join a delegation certificate, issued to a principal represented by its name, with an identity certificate, issued by a trusted certification authority. Another way (shown in Figure 4) is to issue a delegation certificate directly to a principal represented by its public key. In both cases, each principal will use its private key to sign its access requests.

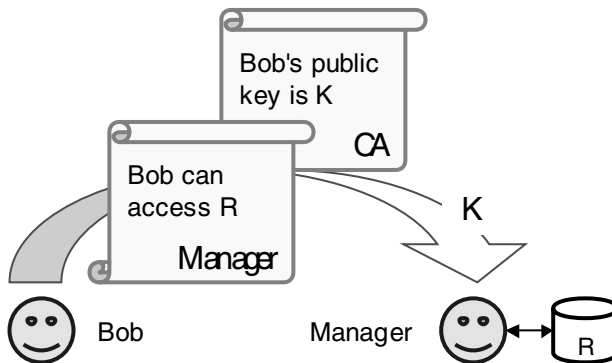


Fig. 3. Identity-based access control

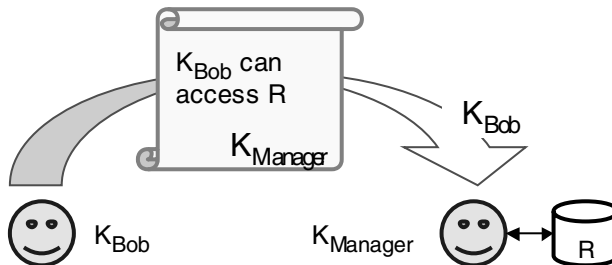


Fig. 4. Key-based access control

The main issue about the first approach is that it requires a certification authority (a trusted third party) to sign identity certificates, so there are two issuer keys that can be

potentially subverted. If, instead, authorizations are directly issued to keys, then there's only one authority and one key to protect.

Another concern is about names as linkage between certificates. The authorization certificate must be issued to a name defined by a certification authority, so the first key has to use a foreign (global) name space and to make a guess about what a name means. This guess is subject to mistakes and attacks, as different principals may have similar names in distinct namespaces.

If, however, the same key, in its own local name space, issues both certificates, then above concerns don't apply. But performance issues remain, about the burden of signing, sending, storing and verifying one more certificate per delegation. These issues can be superseded only if names are really useful at the access control level. Otherwise Occam's razor applies.

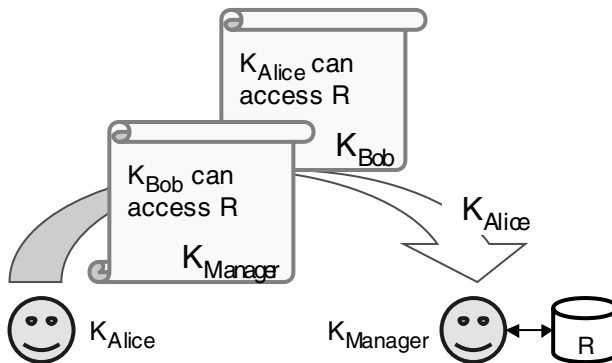


Fig. 5. Access control with certificate chains

When a principal requests access to a protected resource, it attaches a complete certificate chain and a signature to its request message. The resource manager will first authenticate the request and check each certificate. Expired or tampered certificates will be promptly discarded.

The final set of permissions granted by the chain will be evaluated as the intersection of the set of permissions granted to the first issuer (this set could be read in an access control list) with every set of permissions authorized by single certificates. In the same way, the final validity period of the chain will be evaluated as the intersection of validity periods defined for single certificates.

In particular, the resource manager will verify that:

1. first certificate is issued by a known manager of the resource;
2. each certificate is issued by the subject of previous certificate;
3. last certificate is issued to the principal that is making the request;
4. required permissions are listed in each certificate.

It's important to underline that, as every principal can sign its own certificates, one could delegate more permissions than it really has. Thus the final set of permissions can be safely evaluated only by intersecting the sets of permissions carried by each certificate.

Figure 5 shows a principal sending a request to a resource manager. It's worth noting that each principal involved in the chain is directly represented by its public key and not by its name.

If a number of permissions are needed to access a resource, then different certificate chains can be attached to the request: the set of granted rights will be evaluated as the union of rights that flow through each individual delegation chain.

3.3 Local Names, Roles, and Domains

Even if access control can be implemented relying only on authorization certificates and public keys, this doesn't imply names should be avoided altogether. People are used to manage names in several situations, and they prefer dealing with names more than cryptographic keys, even while defining security policies. But names that people habitually use are not globally unique names. These names are rather local names, and need to be unique only for the person or the organization that defines them.

Even if local names are defined in the namespace of their principal (i.e. a public key), they can be useful to others, too. Indeed, in [3] authors show that a local name can be managed as a global name, if both the public key and the identifier are listed explicitly, without relying on a globally trusted public certification authority or a global directory of names. For example *K1 Alice* and *K2 Bob* are proper names defined by principal *K1* and principal *K2*, respectively. A more explicit syntax for names could be *K1's Alice*, as to emphasize that the identifier *Alice* is the one defined by *K1*, precisely.

Local namespaces can also be reciprocally linked, by means of *extended names*. Extended names consist of a principal followed by two or more identifiers. Example of extended names are *K1 Alice Bob* or *K2 Bob Carol*, referring to the entity defined *Bob* by the entity defined *Alice* by principal *K1*, or to the entity defined *Carol* by the entity defined *Bob* by principal *K2*, respectively.

Principals are allowed to export the names they defined, by signing name certificates. A name certificate binds a local identifier to a subject expressing the meaning intended for that name. Public keys, simple and extended names are all legal subjects of name certificates.

So, a certificate can link a name to a public key, better separating the name one uses to refer to a principal, from the key a principal uses. Also, having different certificates that bind a single name to a number of different keys is perfectly legal. Each issued certificate defines a key as a valid meaning for the defined name. One can easily assume that each key is a member of a named group. Given that a name certificate can link a name to another name, then defining complex hierarchies of names, for example to represent roles and domains, is simply a matter of issuing appropriate certificates.

A desirable feature, especially useful when administering a complex platform with a large number of users, is the ability to grant a set of permissions to a *role*. If local names are used, then it's easy to have roles organized as groups, where principals can be added simply by issuing a name certificate. If a principal plays a role (i.e. it is a

member of a group), then it will be granted all permissions intended for that role. Having such a hierarchy of principals in place, then each user can represent a parent node for all his agents and containers, or each container can represent a parent node for the agents it hosts.

If a hierarchical organization of principals proves useful, the same is valid for resources. In large complex systems, the ability to delegate responsibilities to manage a group of resources to a *domain* administrator is a common issue. Domains are especially useful when defining target of permissions, as they name an entire set of resources within a single permission. If agents are organized in a hierarchy of named groups, then a principal could obtain the right to manage an entire group of them, for example all agents owned by *Bob*, or all agents hosted on *Container1*, or even all agents hosted on a container owned by *Alice*. Each principal can define its own namespace, so each entity that controls some resources can define its own named groups of resources. As a rule of thumb, permissions should ever be expressed using the namespace of the principal eventually responsible for access control, so the authorizer never needs to rely on external naming authorities.

3.4 Trust Management Principles

Till this point, we described means to enforce a configurable policy on the overall agent platform. However, looking at the model defined at the beginning of this paper, we can see it as composed of different cooperating components.

Following the approach of [7], this system can be described as a community of peers, each one able to play the role of a controller or a requester. If an entity ultimately controls access to resources, being able to authorize or refuse requests for their usage, it plays the role of a controller; if an entity requests to access resources controlled by other entities, it plays the role of a requester.

To have a sound system, all these peers should adhere rules of *trust management*. In [6] these rules are summarized as:

1. ***be specific***: ‘Alice trusts Bob’ is a too vague concept; it has to be better quantified in expressions as ‘Alice trusts Bob to read file.txt in her home directory today’;
2. ***trust yourself***: all trust decisions should be founded on sound, local believes; when possible, trusted third parties should be avoided, especially if their mechanisms and politics are not known;
3. ***be careful***: even the best implementation can be violated if users behave superficially and expose reserved data.

Applying these rules requires each component to be described as an authority, responsible for protecting its local resources and for managing its trust relations. This *modus operandi* will provide trust relations among platform components a better grounding, and the possibility to define policies both at the container level and at the agent level.

This applies to agents that, on one side, can define application specific permissions and issue certificates for granting them to trusted partners and, on the other side, can

use their own private key to sign requests, thus being acknowledged by other resource managers.

But this applies to containers, too. Indeed containers have to protect resources of the underlying operating system, like files and network connections, as well as the agents they host. This resources need to be protected from threats posed by external (to the container) entities, or even internal entities, i.e. hosted agents.

Finally, organizing a platform as a community of peers connected by trust relations, allows each component to easily adapt to larger environments, where multiple platforms interact. And relying only on public keys helps keeping things simple. An agent can seamlessly sign authorization certificates for its neighbors, or for agents living on different platforms. Indeed, both local and remote agents are simply identified as keys.

In addition, dealing with public keys, only, paves the way for a two-layer security infrastructure. Under a higher level, where interconnections among agents and containers are handled and secured, a lower level can be sketched, where generic security means are provided to allow distributed trust management. This layer can be clearly separated from agent related packages, thus being useful in traditional applications, too. Sharing a common low-level security infrastructure, where key-based authorities take charge of distributed management of access rights, allows agents and components based on different technologies to interoperate without weakening resource protection.

4 Security in JADE

As a concrete outcome of addressing the security and delegation issues in multi-agent systems, this paper presents a case study about the design and implementation of multi-user and security support for JADE, a software framework to develop multi-agent systems in compliance with the FIPA specifications. JADE supports most of the infrastructure-related FIPA specifications, like transport protocols, message encoding, and white and yellow pages agents. Moreover it has various tools that ease agent debugging and management.

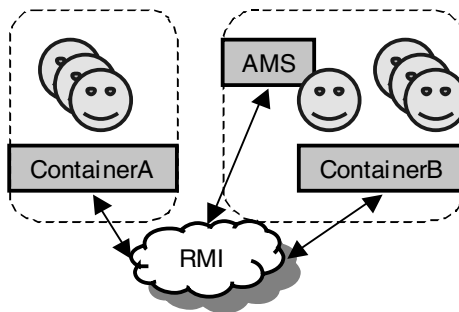


Fig. 6. Current JADE platform

However, no form of security was built into the JADE agent platform before version 2.61, and the system itself was a single-user system, where all agents belonged to a single owner and had equal rights and permissions. This means that it was not possible to use JADE in several real world applications, such as electronic commerce.

The architecture of a JADE system, shown in Figure 6, is centered on the concept of platform, which is essentially a federation of agent containers possibly distributed across multiple hosts. Apart from federated containers, each platform includes other components, as the AMS (Agent Management System) and the DF (Directory Facilitator) described by FIPA specifications.

In particular, the AMS runs a white pages service; agents can reach it through ACL messages to register or deregister themselves, and to search for other agents; it is notified by containers about relevant events, as births and deaths of their hosted agents; the AMS, on its turn, can contact each container when it needs to manage (create, kill, suspend, resume) the hosted agents. We didn't explicitly draw it in our generic model because, even if some systems, as JADE, include it as a separate component, it's not the general case. In some systems, and maybe even in future version of JADE, the functionalities of the AMS could be carried on by the containers themselves or by the underlying network infrastructure, as suggested in Figure 7.

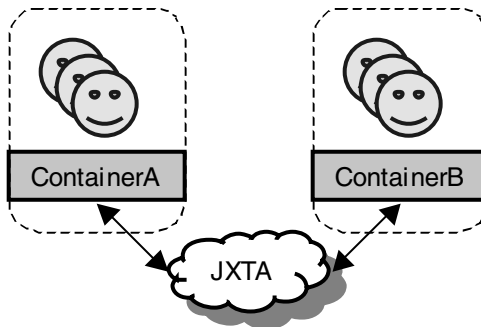


Fig. 7. P2P-based JADE platform

The DF, instead, is not directly involved in the management of the platform and it can be considered as an application-level service. In fact in JADE its duties are brought on by an agent or by a federation of agents hosted on the platform.

Securing an agent platform implies all hosted resources listed into the previous section have to be protected, including reserved data, agents, containers, files. Resources have to be protected from both external and internal threats, preventing external entities from damaging the platform in any way and preventing agents from harming other agents and from accessing resources without required authorizations.

So, efforts have been directed toward:

1. securing communications among infrastructure-level components, i.e. among containers and with the AMS; this implies reciprocal authentication of connected parties, to assure only trusted components can join the platform, as well as integrity and confidentiality of exchanged data;

2. forcing agents to adhere to the defined policies; this requires each agent to be associated with an authenticable principal and with a certain set of granted permissions; means must be provided for agents to delegate permissions to each other, too.

As the first point concerns, a number of existing technologies are designed to assure protection of transmissions on public networks, among which the SSL (Secure Sockets Layer) protocol is emerged as the standard for secure communications on the Internet. SSL is designed to secure general-purpose network connections, and it can guarantee the integrity and confidentiality of TCP connections. It allows mutual authentication of both parties in a network connection, too. This feature allows a container to protect itself from intrusions, preventing malicious applications from masquerading as trusted components of the platform. Moreover, as SSL is placed at the socket level, it can be easily inserted in an existing network infrastructure, as network security is encapsulated in a very low level and its details remain hidden in that level.

The security model that is included in JADE from version 2.61 focused on the protection of a platform from malicious actions taken by external entities or by hosted agents and some simplifications were adopted to address these specific threats: in particular only one authority and one couple of cryptographic keys was in place in each platform, associated with the AMS, and so the responsibility to define the security policy was strongly centralized, even if agents and containers could access the authority to ask for delegation certificates that later they could distribute to their trusted partners.

In the following subsections, we will present a generalized model where multiple entities, including agents and containers, can sign and distribute their own certificates.

4.1 Principals, Resources, and Permissions

In our system, a principal is any entity that can take actions and can be held responsible for them. Agents, containers and the AMS are certainly to be considered principals. Users cannot directly perform actions on the platform, but they take responsibility for the actions performed by their own agents and containers on their behalf. So, in a JADE system, users are represented as principals, too. Even external entities, given they are able to access the platform and take some actions on it, for example leveraging on available message transport protocols, should be considered principals.

Resources needing access protection in multi agent systems certainly include all resources of underlying environments, as file systems and network connections. These resources must be protected from unauthorized access, leveraging on existing security frameworks when feasible. But multi-agent systems have to protect their agents and their infrastructures, too.

Remote containers must protect themselves from reciprocal threats. Unauthorized actions could include suspending or killing an agent, routing a false message, closing the container. Agents themselves could represent threats to their containers, especially when agent mobility is allowed. In many cases the running environment is based on

some kind of reference monitor, so agent actions can be controlled and limited; but denial of service and other subtle attacks are difficult to prevent.

In their turn, agents must have a strong trust toward their hosting containers, as they have no means to prevent a container from stealing their code and data, slowing or stopping their execution. Only after the fact actions can be taken, assuming that a detailed log of system events could be traced.

Permissions express which actions can be taken on which resources. Typical permissions include target identification, and a list of actions allowed on that target; both targets and actions are domain dependent. Permissions are usually stored in policy files and access control lists, where each known principal is bound to a set of granted permissions.

In multi agent systems, proper permissions must be at hand to represent actions that are specific to their particular context. These permissions should list (groups of) agents or containers as targets, and then actions allowed on them. JADE adds to the large set of permissions defined by Java other permissions to describe actions on agents and containers. Actions on agents, that can be listed in an *AgentPermission* object, include delivering messages, suspending, resuming, creating, killing, moving and cloning. Containers can be created, killed, or asked to host new agents, copy or clone them and these actions can be listed in a *ContainerPermission* object.

Agents could want to define their own permissions, too. These permissions should protect application level resources, as inner data structures, network connections, physical devices, files and databases managed by the application.

4.2 User Authentication

Agents and containers are bound to users at creation time. JADE becomes a multi-user system, similarly to modern operating systems, and users 'own' agents and containers. This binding is managed by the platform and the user is prompted for classical username and password. It is also possible to subsequently change the ownership, if that is permitted by the platform policy, and username and password are asked again. JADE containers have a password file against which passwords are checked. As in many other systems, hashes are stored instead of clear text passwords.

Agents can use the permissions defined by the platform policy and, as mentioned above, also permissions delegated from others. Actually the agent keeps a set of certificates into its own *CertificateFolder*. All the delegation certificates passed by others are kept into this folder. At creation time, just after the validation of username and password, the container authority takes all the permissions contained into the local policy file and referring to that user. Then creates a delegation certificate containing all those permissions and adds it to the certificate folder of the agent. This certificate can be used by the agent like any other delegation. This way the system delegates at creation time the permissions defined by the policy file.

4.3 Certificate Encoding

Various specifications have been proposed to encode authorization certificates. For example, the original SPKI proposal [3] suggested to encode every object using s-expressions. But nowadays a better choice could be XML, allowing the exploitation of existing tools to encode and decode certificates. Moreover, adopting XML greatly ease the work of signing the certificates, as XML-DSIG is winning more and more consent from the developers' community, and good tools are beginning to appear. An example of an XML-encoded certificate is presented in Figure 8, following some of the guidelines proposed in [8].

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/01/xmldsig/">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/..."/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/01/xmldsig/dsa"/>
    <Reference IDREF="auth-cert_1234">
      <Transforms><Transform Algorithm="http://www.w3.org/..."/></Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/01/xmldsig/sha1"/>
      <DigestValue Encoding="http://www.w3.org/2000/01/xmldsig/base64">
        oYh7..TZQ0dQ0dmQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>gYFdrSDdvMCwCFHWh6R...whTN1k==</SignatureValue>
</dsig:Object Id="auth-cert_1234" xmlns="http://www.w3.org/2000/01/xmldsig/">
  <cert>
    <issuer><public-key><dsa-pubkey><dsa-p>ewr3425AP...1yrv8iIDG</dsa-p>...
    </dsa-pubkey></public-key></issuer>
    <subject><hash-of-key><hash hash-alg="sha1">
      AMmGTeQjk65b82Jggdp+0A5MOMo=</hash></hash-of-key></subject>
    <delegation/>
    <tag><set>
      <agent-permission>
        <agent-name>agt1@platform</agent-name>
        <agent-action><set>create kill</set></agent-action>
      </agent-permission>
      <agent-permission>
        <agent-name>agt2@platform</agent-name>
        <agent-action><set>send receive</set></agent-action>
      </agent-permission>
    </set></tag>
    <validity><notbefore>2003-04-15_00:00:00</notbefore>
    <notafter>2003-04-18_00:00:00</notafter></validity>
    <comment>Subject can create/kill agt1 and communicate with agt2</comment>
  </cert>
</dsig:Object>
</Signature>
```

Fig. 8. An XML-encoded authorization certificate

4.4 Access Control

Once got a delegation of a JADE permission, an agent can perform the permitted actions just like it always had the permission to do so. For example, once received a delegation for moving from a container to another, the agent can just move as would normally do. Actually, behind the scene, the container authority carefully checks all the times that the action is really allowed, looking at the certificate folder of the agent.

First, all the certificates in the certificate folder are verified. This is performed locally by the container hosting the agent, since certificates carry all the information needed to complete the task. At this point a new Java protection domain is created with all the delegated permissions, and bound to the executing thread. Our approach mimics the JAAS [12] principal-based access control mechanism but extends it to deal with the delegation of policy responsibilities among trusted components.

In fact, while JAAS assume that policy resides in a local file and all policy decisions can be defined locally, this is not always possible in distributed and evolving environments. Moreover, in multi-agent systems, principals are not necessarily bound to a fixed set of permissions, as an agent can play different roles at different times, exploiting only a subset of the certificates and the permissions it has been delegated.

5 Conclusions

Systems based on distributed models such as multi-agent-systems (MAS) and peer-to-peer systems (P2P) need effective security techniques in order to provide real-world solutions. Traditional frameworks take advantage of cryptography techniques, certificate authorities (CA) and access control lists (ACLs) for granting levels of trust amongst peers and users. Today, extremely dynamic and large systems pose a big challenge in terms of manageability, scalability and robustness.

In this paper we discussed our implementation of security mechanisms for JADE, a framework for distributed multi-agent systems. Moreover, we pointed out some concepts that might bring security services further toward largely distributed agent-based systems where dynamic trusted relations building, collaboration and delegation of duties become regular activity of many entities into a large-scale system. The adoption of concepts from SPKI and trust management principles led us to a model where policies are based on keys instead of names. Authorization does not need certificate directories for binding names to public keys, thus problems about unavailability of such services do not have negative impact on the whole system. This enhances scalability and robustness for large and dynamic systems, avoiding centralized authorities or any other potential bottleneck, without sacrificing security.

References

1. Poggi, A., Rimassa, G., Tomaiuolo, M. Multi-user and security support for multi-agent systems. Proc. WOA 2001 (Modena IT, September 2001), 13–18.
2. Vitaglione, G. JADE Tutorial - Security Administrator Guide. <http://jade.cselt.it/doc/tutorials/SecurityAdminGuide.pdf>. September 2002.
3. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T. SPKI certificate theory. IETF RFC 2693, September 1999.
4. Aura, T. On the structure of delegation networks. Proc. 11th IEEE Computer Security Foundations Workshop (Rockport MA, June 1998), 14–26.

5. Blaze, M., Feigenbaum, J., Lacy, J. Decentralized trust management. Proc. 1996 IEEE Symposium on Security and Privacy (Oakland CA, May 1996). 164–173.
6. Khare, R., Rifkin, A. Weaving a web of trust. World Wide Web Journal, 2, 3 (Summer 1997), 77–112.
7. Li, N., Grosz, B. A practically implementable and tractable delegation logic. Proc. 2000 IEEE Symposium on Security and Privacy (Oakland CA, May 2000), 29–44.
8. Paajarvi, J. XML Encoding of SPKI Certificates. Internet Draft draft-paajarvi-xml-spki-cert-00.txt, March 2000.
9. Jansen, W., Karygiannis, T. Mobile agent security. NIST Special Publ. 800–19.
10. FIPA. <http://www.fipa.org>.
11. JADE. <http://jade.cselt.it>.
12. JAAS. <http://java.sun.com/products/jaas/>.

A Conceptual Framework for Web Services Semantic Discovery

Islam Elgedawy

Computer Science Department, RMIT University, Australia
elgedawy@cs.rmit.edu.au

Abstract. Current service discovery techniques mainly depend on keyword matching approaches. That could lead to a high service retrieval recall whereas it leads to a low service retrieval precision. To overcome this problem, a service matchmaking process should use the semantics of both the service and the user. Hence, we propose the G^+ conceptual model that explicitly captures the service/user objectives, scenarios and contexts. Scenarios capture the required snapshots of the service/user behaviour. Based on the G^+ model, we introduce the PILLAR conceptual model that captures multiple semantic descriptions of the service. The PILLAR concept is also used to semantically model the user request. Different domain ontologies could be used to provide a service/user PILLAR models with the basic vocabulary. Consequently the ontology mapping process becomes fundamental during the matching process. Hence, we propose an ontology meta-structure to facilitate the ontology mapping process. PILLAR explicitly captures the semantics of both the service and the user creating a framework for more precise service semantic discovery process.

1 Introduction

In the service discovery process, the services are located without a-priori knowledge. The service provider has to register his services into a services directory where a user submits his query. After that the user contacts the service provider to support him with the required services. UDDI (www.uddi.org) is a standardized specification for universal services description, discovery and integration that provides a service registry for web services. Unfortunately, UDDI search mechanism relies on pre-defined categorizations through keywords matching which leads to unaccepted low retrieval precision [1][18].

The newly emerging technologies and service discovery protocols, such as Service Location Protocol (SLP) [7], Salutation [21], JINI [9], Simple Service Discovery Protocol (SSDP) [16], and Secure Service Discovery Service (SSDS) [6] rely on pre-defined or dynamic templates for modelling the services. Templates' attributes definitions rely on text description and pre-defined flag values to describe the attribute semantics, which is not rich enough to define the service functionality. Hence, the service matchmaking process is effectively done at the syntactic level, whereas it is inefficient for the current heterogeneous scalable environments [2][14].

To increase the service discovery precision, the service matchmaking process should be based on Web services semantics such as service functionality [18]. The

lack of a common flexible conceptual model for representing the semantics of the Web services leads to a semantic interoperability problem during the service match-making process that services have same functionality but have different representations will not match [13][14] [15].

DAML_S [1] was introduced as a common semantic model for Web services. DAML_S consists of a service profile that describes what the service does, a process model that describes how the service works and finally a service grounding that describes how the service is used. Unfortunately, DAML_S profile captures the service functionality as a text description; hence keywords are still used for matching. Also DAML_S does not support user-service mediation that leads to a low retrieval precision because the user context is ignored, which is necessary for precise discovery process [10].

For more precise service discovery process, the semantics of both the service and the user should be involved during the matching process such as the required goals, the expected behaviour and the involved contexts. We introduce the G^+ model to capture the service/user objectives, scenarios and contexts in a specific domain by coupling these concepts together. A Web service could support different roles in different domains achieving different goals within different contexts; hence we introduce the PILLAR model that adopts the G^+ model to capture these different semantic descriptions of a Web service. PILLAR's concept is also used to semantically model the user request. For total flexibility, different domain ontologies could be used to provide the service/user PILLAR models with the basic vocabulary. Consequently the ontology mapping process becomes fundamental during the matching process. To facilitate the ontology mapping process, we define an ontology meta-structure that defines the basic required ontology components for semantic discovery process.

In section 2, we give a background that addresses the required characteristics for the semantic discovery process. In section 3 we present the ontology meta-structure. In section 4, we introduce the G^+ model. In section 5, we introduce the PILLAR model. In section 6, we give a discussion about PILLAR comparing it with some existing solutions. In section 7, we give conclude our work indicating the future work directions.

2 Background

For more precise semantic discovery process, we believe that the following issues must be supported:

- Use of domain ontology for modelling the service description model and user request that will minimize the semantic dissimilarities between them, which helps to overcome the semantic interoperability problem [4] [15].
- Differentiate between advertising for systems (machine-understandable) and Advertising only for human (not machine-understandable). Machine-Understandable description is used during the automatic service discovery process. We define this service description as the "*service active description*". While non machine-understandable advertisement will not participate in the automatic service discovery process. The human user has to judge the service according to his understanding. We define this description as the "*service pas-*

sive description". The service discovery technique will differ according to the available service descriptions (active or passive or both).

- Having an explicit service/user goal model would facilitate the semantic discovery process by applying a goal discovery phase that filters the services with different goals, avoiding the use of keywords for matching. Having a goal model facilitates automatic service composition (composing a complex service based on the user's goal hierarchy) and justifies the presence of services components that are not necessarily comprehensible to the agents [3].
- Support multiple abstraction levels for the service behaviour will facilitate the agent task for judging the service. The service behaviour should be advertised in a top-down approach. Starting from the most abstract description at the top level and reaching to the most detailed description at the bottom level. So an agent can trace this hierarchy according to its interest.
- Support multiple contexts awareness such as the *goal context* that defines the constraints and conditions that must be hold to achieve the goal. The *technical context* that defines the technical requirements that must be fulfilled that the service or the agent can function such as the required device type, bandwidth, proximity and communication protocol [10].
- Support multiple mediation levels between the user request and the services models such as mediation at the level of business logic, dynamic service invocation, data structures, and message exchange protocols [4]. The mediation paradigm depends on resources and customer needs models, and involves human intervention [23].

3 Ontology Meta-structure

The universe of discourse consists of a collection of domains. Every domain may consist of a collection of sub-domains. Every domain or sub-domain may have different representations, which are realized by different ontologies. The Web services refer to its used ontology(s). To facilitate the ontology mapping process we propose a meta-structure whose elements define the basic required ontology components for semantic discovery process. The meta-structure consists of concepts, operations, rules, and roles (Fig. 1).

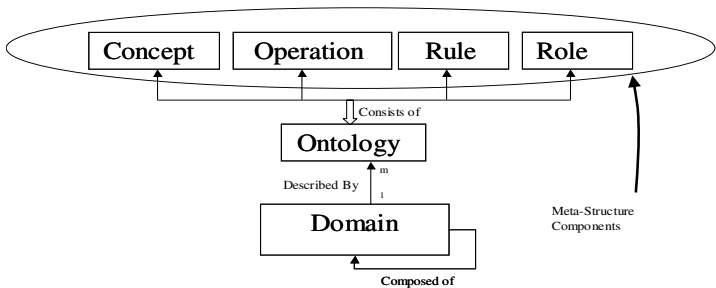


Fig. 1. Relations among domain, ontology, and the meta-structure

The domain entities are described by a set of atomic attributes and values that represent the characteristics of the entity. For example for the flight entity we can have attributes such as id, name, source, destination, price, etc. The total subsumption ($\prec$) and equivalence ($\cong$) relations are fundamental relations during the discovery process because they provide the base to relate concepts with each other and they provide the means to build the domain concept-hierarchy that is used in the match-making process between the services and user request [22]. For example the user asking for a room reservation and the service provides hotels reservations. In this case the subsumption relation between the room and the hotel will be used for matchmaking process.

Concepts describe the domain entities and their relations. For example in the tourism domain we can have entities such as holiday, flight, ticket, hotel, resort, etc. A concept is defined as a tuple of an entity that captures the concept attributes and a set of concept subsumption and equivalence relations. (The model is defined later)

Operations describe the domain legitimate transaction types and their relations. For example in the tourism domain we can have operations such as hotel reservation, flight booking, money payment, etc. An operation is defined as a tuple of an entity that captures the operation attributes, a set of operation subsumption and equivalence relations, as set of input concepts and a set of output concepts. (The model is defined later)

Roles describe the domain legitimate actors and their relations. For example in the tourism domain we can have roles such as customer, travel agent, hotel manager, etc. A role is defined as a tuple of an entity that captures the role attributes and a set of role subsumption and equivalence relations. (The model is defined later)

Rules describe the domain legitimate derivations and the rule bases that are based on the defined concepts, processes, and roles. For example in the tourism domain we can define some common rules for VIP people. We define the rule as a mapping from a rule domain to a rule range. (The model is defined later)

We define the domain ontology as a tuple of a set of concepts, a set of operations, a set of roles and a set of rules. A domain is defined as a tuple of an entity that captures the domain attributes and a set of ontologies that represent that domain. Both the user and the service provider should explicitly indicate which ontology used as a basis for his terminology.

The Model for the meta-ontology is defined as:

$Value = Numeric \mid String$

$Attribute = "identifier"$

$Entity = \{ \langle attribute, Value \rangle \}$

$Concept = \langle Entity, \{Crelation\} \rangle$

$Crelation = " \prec " \langle Concept, Concept \rangle \mid " \cong " \langle Concept, Concept \rangle$

$Operation = \langle Entity, \{Orelation\}, \{Input\}, \{Output\} \rangle$

$Input = Concept$

$Output = Concept$

$Orelation = " \prec " \langle Operation, Operation \rangle \mid " \cong " \langle Operation, Operation \rangle$

$$Role = \langle Entity, \{ Rrelation \} \rangle$$

$$Relation = " \prec " \langle Role, Role \rangle \mid " \cong " \langle Role, Role \rangle$$

$$Rule: Concept \times Role \times Operation \rightarrow Concept \mid Role \mid Operation \mid Value$$

$$Ontology = \langle \{ Concept \}, \{ Operation \}, \{ Role \}, \{ Rule \} \rangle$$

$$Domain = Domain \mid \langle Entity, \{ Ontology \} \rangle$$

4 G^+ : Extended Goal Model

A service/user goal model should be rich enough to represent the service/user objectives, since a goal is intentional by nature; it lacks of the operational point of view leading to a limited requirement representation. Research in the requirement-engineering (RE) field proposes a solution for this problem by coupling the goal and the scenario concepts together [11][19][20].

Our vision is to have a full-fledged goal model that represents service/user goals, scenarios and contexts. Unfortunately, there is no current model that couples these concepts together. To realize our vision we need to answer the following questions: How can domain ontology be used to unify the terminology? How can a goal be modeled? How can a context be modeled? How can a scenario be modeled?

Domain ontology will provide a user/service provider with concepts and operations definitions that are used to compose the required goals, scenarios and contexts.

A goal may not be clear enough for representing all the required objectives; hence more detailed levels of sub-goals would be created until a clear objectives representation is reached. As a goal can be expressed as a clause with a main verb and several parameters [20] we model a goal as a hierarchy of operations with their related concepts. Hence, every primitive sub-goal is described by an operation that is defined in the domain ontology because an operation represent the action described by the clause's main verb.

As scenarios are sufficient to express a majority of system's behaviours that are necessary for the purpose of scenario based goal modelling [20]. Every goal and sub-goal will be attached to a collection of different scenarios that realize it, textual scenarios are recommended because they are easier to process [19]. Scenario is defined as an execution of sequence of operations [11][20]; hence a scenario could be modeled as a graph that has a start node and finish node. The node represents the operation and it corresponding concepts. And the edges represent the sequence of nodes execution.

Every goal and sub-goal has a specific context that defines its constraints. Unfortunately, there is no common context model; hence we adopt the trend that models the context as a set of rules [5] because the rules could be realized as the pre-conditions and post-conditions for the service/user goal, which facilitates the matching process.

As the above questions are answered, we define the G^+ extended conceptual goal model as in Fig. 2. The goal is composed from sub-goals. Many alternative scenarios could realize every goal/sub-goal. Any types of scenarios could be coupled with the goal such as exceptional handling scenarios. Every goal/sub-goal is described by an operation. The goal/sub-goal context is modeled as pre and post conditions over a set

of defined concepts. The scenario consists of many operations that could be realized by more detailed scenarios.

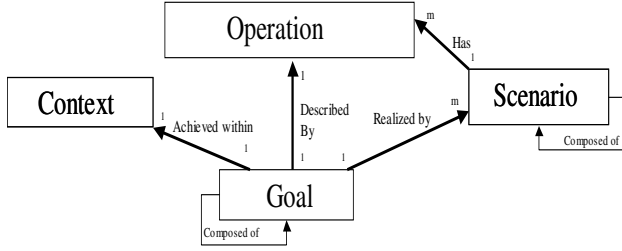


Fig. 2. The G^+ Conceptual Model

The G^+ Model is defined as follows:

$$\begin{aligned}
 \text{Scenario} &= \langle \text{Start}, \{\text{Node}\}, \{\text{Edge}\}, \text{Finish} \rangle \\
 \text{Edge} &= \langle \text{Node}, \text{Node} \rangle \\
 \text{Start} &= \text{Node} \\
 \text{Finish} &= \text{Node} \\
 \text{Node} &= \langle \text{Operation}, \text{Detail} \rangle \\
 \text{Detail} &= \text{Scenario} \mid \text{empty} \\
 \text{Context} &= \langle \{ \text{Concept} \}, \{ \text{Pre-Condition} \}, \{ \text{Post-Condition} \} \rangle \\
 \text{Pre-Condition} &= \text{Rule} \\
 \text{Post-Condition} &= \text{Rule} \\
 \text{Goal} &= \text{Goal} \mid \text{Operation} \\
 G^+ &= G^+ \mid \langle \text{Goal}, \{\text{Scenario}\}, \text{Context} \rangle
 \end{aligned}$$

The following examples indicate how to build the user/service G^+ models.

Example 1

Suppose a user lives in Melbourne and wants to book a holiday to Cairo during the month of February 2003. To accomplish this goal he wants to book a flight and make a hotel reservation. He wants to use his VISA credit card and not to pay more than \$2000 for the flight and no more than \$500/week for the hotel. Fig. 3 indicates how the G^+ is built. In this example, the user goal is to book a holiday. To achieve that there are two sub-goals are required book a flight and book a hotel room. The context attached to a goal or a sub-goal adds more information that explains the required objectives such as the user wants to book a room with two beds in the Hilton. The “book hotel room” goal, the goal is considered achieved if and only if a room with two beds in Hilton with price less than or equal to \$500/week is booked hence the context could be defined as a desired post conditions of the goal. Also we notice the attached scenarios explain the user required rational for achieving these goals such as he wants to request a ticket and pay the money with his credit card and receive a confirmation in order to book a flight.

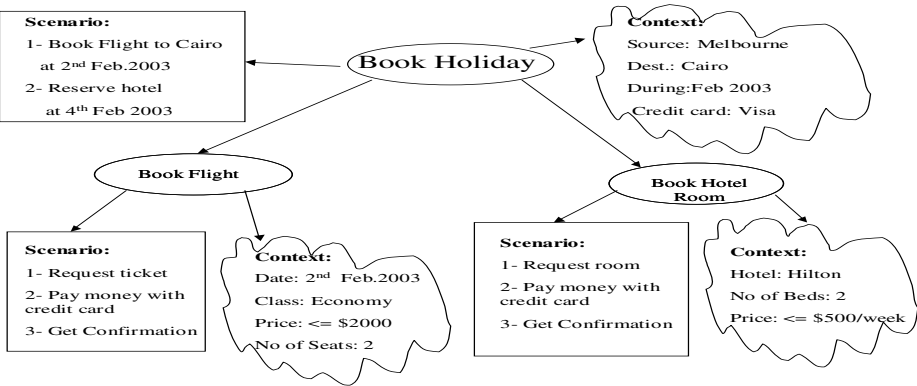


Fig. 3. The user G^+ model

Example 2

Suppose a travel agent in Egypt who organizes holidays. He/She can book the flights and hotels and can negotiate the prices and arrange tours. His/Her service will be advertised as indicated in Fig. 4. In this example the travel agent service has many advertised goals; reserve flight, organize tours and reserve hotel rooms. Every goal is attached with its context and the describing scenario. The “Reserve flight” goal can only be fulfilled if the ticket destination is Egypt and realized as requesting a ticket then paying the money and at the end the provider will send a confirmation to the requester. The “reserve flight” goal is achievable if and only if the destination is Egypt hence it could be defined as the goal pre condition

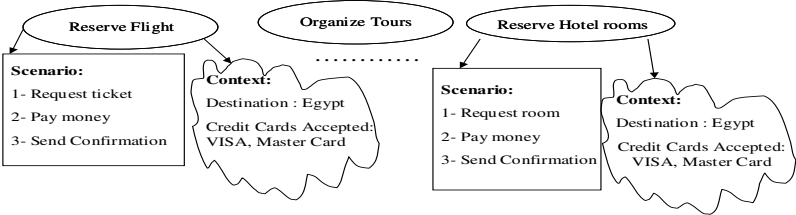


Fig. 4. The service G^+ model

5 PILLAR Model

As the service model should explicitly differentiates between the service active description (machine-understandable) and the service passive description (not machine-understandable). As in Fig 5, we represent the service active description by a service active view (SAV) that captures the service domain, the used ontology, the supported domain roles, the G^+ model(s), and the service technical context. The service passive description is represented by a service passive view (SPV) that captures the service deployment model and the service business documents.

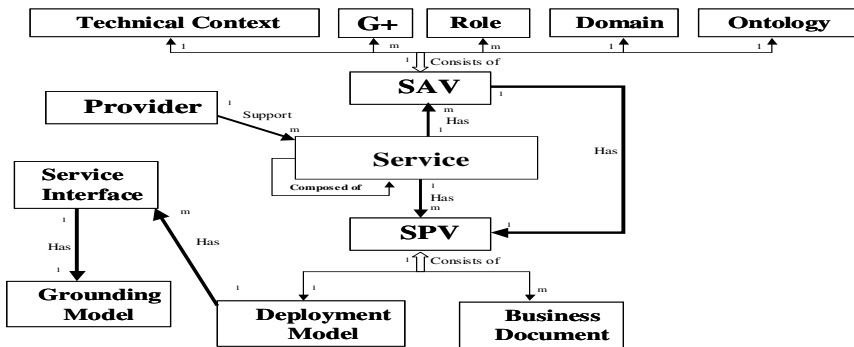


Fig. 5. The service PILLAR model

SAV could have many G^+ models that could be achieved within the same domain and the same technical context. As in example 2, the service has many goals could be achieved in the tourism domain. A Web service could have many SAVs to advertise itself in different domains using different ontologies achieving different goals within different technical contexts. For example a hotel could advertise its service in both tourism and business domains. Every domain has its corresponding goals. In tourism domain, goals could be as “reserving rooms” while in business domain, goals could be as “organizing meetings”.

Every SAV has a SPV that contains the corresponding deployment model and the corresponding business documents such as UML diagrams. The deployment model consists of the service interfaces that suit the advertised SAV. For example, if the service supports both PDAs and PCs then two SAVs will be created. Every SAV has a different technical context that supports the corresponding device type. The corresponding SPV will provide the user with the corresponding service interfaces and business documents that suits the device type. The human user could use the SPV to have a meticulous judgment over the service capabilities.

Every service interface has the corresponding grounding model that indicated how this interface would be used. For example if the service will be executed over a SOAP protocol, the interface could be described using WSDL that support the suitable binding for SOAP [1].

A Web service could have special attributes that describe its entity such as ID and name. Hence we define a service header that holds the provider’s information and any special attributes defined by the provider.

A Web service could be complex that it can use other Web services to accomplish some of its tasks. For example a credit card verification service could be invoked by the service depicted in example 2. Hence, the service could be modeled as a collection of other Web services. Which enhances the Web service dependability that if a sub-service is failed, a replacement service could be discovered and invoked to recover the failure. Even more a Web service could define a list of emergency services that could be invoked if one of its current sub-services has failed.

The service **PILLAR model** is defined as follows:

$Header = \langle Entity, Provider \rangle$

$Provider = Entity$

$SAV = \langle Technical\ Context, \{ G^+ \}, \{ Role \}, Domain, Ontology \rangle$

$Technical\ Context = Context$

$SPV = \langle \{ Business\ Document \}, Deployment\ Model \rangle$

$Business\ Document = Entity$

$Deployment\ Model = \{ Interface \}$

$Interface = Operation$

$Service = Service | \langle Header, \{ \langle SAV, SPV \rangle \} \rangle$

The PILLAR's concept could be applied for modelling the user request. A user will have an only active view that participates in the discovery process. The user active view (UAV) will consist of the user domain, the used ontology, the user's role, the G^+ model, and the user technical context (see Fig. 6). The use of UAV during the semantic discovery process will be discussed in the next section.

The user **PILLAR model** is defined as follows:

$User = \{ UAV \}$

$UAV = \langle Technical\ Context, G^+, Role, Domain, Ontology \rangle$

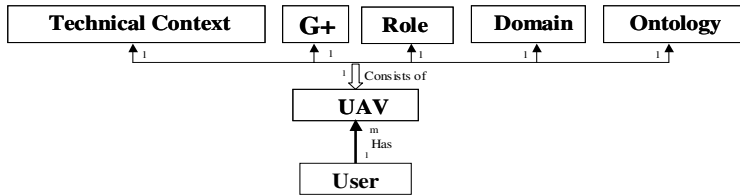


Fig. 6. The user PILLAR model

6 Discussion

A comparison between PILLAR and different service models addressed in different solutions is summarized in table 1. These solutions are discussed within many different fields like agents, Web computing and business. We define four general scales to evaluate the support of a characteristic: (F) means fully supported that the characteristic is discussed and models and solutions are introduced. (P) means partially supported that the characteristic is discussed but neither solutions nor models are proposed. (S) means similarly supported that similar concepts are addressed. (N) means the characteristic is not supported.

DAML_S [1] partially supports the use of domain ontology because the “service category” parameter in a DAML_S service profile refers to the used domain ontology, but there is no explanasion about how the onology will be used to define the vocabulary. DAML_S does not have an explicit definition for the context concept but it defines the preconditions for the service that could be considered “similar” to the context definition in PILLAR. DAML_S describes the service process model to capture the service behaviour that could be considered similar to the scenario model. Finally the DAML_S service profile describes the service description in a non machine-understandable format.

UDDI only support some text descriptions for the service functionality and the service provider in a non machine-understandable format, which is similar to the PILLAR passive description. UDDI does not suport any service deployment models.

Table 1. Comparison between PILLAR and different service models addressed in different solutions

Model	PILLAR	DAML_S	UDDI	LARKS	ebXML	WSMF	espeak
Characteristic							
Use of Domain Ontology	F	P	N	S	N	P	S
Goal Model Support	F	N	N	N	N	F	N
Context Model Support	F	S	N	F	N	S	N
Scenario Model Support	F	S	N	N	N	S	N
Active Description Support	F	N	N	S	N	S	S
Passive Description Support	F	F	S	N	F	N	N

F: Fully Supported; P: Partially Supported; S: Similarly supported; N: Not Supported

LARKS [22] does not explicitly define the concept of ontology but they define a set of words that used as a vocabulary basis, which is “similar” to the ontology usage but it lacks of other ontology components definitions. LARKS defines a slot structure to describe the agent capability and use it while the matchmaking process. Which could be considered similar to the active description of the service. LARKS models the context as a set of concepts that are defined by a concept language, and defines the pre/post conditions for agent invocation. In a nutshell, it is the same trend taken in PILLAR to model the context.

ebXML [17] presents the service as two views, the business operational view(BOV) and functional service view (FSV). The BOV address any semantics description for the service using UML diagrams. FSV address the services interfaces and supported protocols. But both BOV and FSV are published for the human user. Hence the BOV and FSV are presenting information about the service passive descriptions.

WSMF [4] is a framework for modelling Web services. It allows the use of ontology but it did not give a detailed description of how this will be realized. WSMF propose a goal text description repository for the services. That all the goals in all domains should be defined in that repository and every service should be mapped to one or more of repository goals. WSMF does not explicitly define the concept of context but it defines the pre and post conditions for the service. WSMF suggests the use of DAML_S process model for describing the service behaviour. The service attributes introduced in WSMF are used to define the service external description as pre-defined parameters, which could be similar to the active description because it could be processed automatically.

eSpeak [8] registers the service as an attribute base specification that defines the service external description, which is similar to the service active description. The attribute is a set of name-value pairs of a pre-defined vocabulary created by the service provider and published for reuse. Which is similar to the ontology concept.

To formalize PILLAR framework, we need to answer the following questions: How does PILLAR help in service semantic discovery? How PILLAR would be useful in the real world?

- The semantic discovery process will mainly be based on comparing UAV with SAVs located in the service directory. The user defines his UAV then submits it to the matchmaker who will filter the services as the following order. First it will filter by UAV domain that services in different domains will be discarded. Then it filters by supporting roles that services that do not support the UAV role will be discarded. Then it filters by G^+ that services have different goals and contexts will be discarded. Finally it filters by technical context that services that do not match the context requirement will be discarded. Technical context components could be dynamic that require calculation at the discovery time to know their values such as proximity and bandwidth. The user can select from the resulting services based on the services interfaces and business documentation. PILLAR provides the basis to solve the scalability problem by semantically cluster the services based on their SAVs. The matchmaker will match the user UAV with the suitable semantic cluster then applies the mentioned filters over the matched cluster.
- PILLAR could be applied to any domain that its entities have an explicit function and explicit method for carrying out this function. For example in bioinformatics domain, we can model the proteins and genes using PILLAR. Any protein or gene could be modeled as a service. The protein or gene function and method of carrying this function could be modeled within the goal, scenario and context of the G^+ model. For example if a protein requires certain enzyme to be able to function correctly this could be modeled as a pre-condition in its G^+ context.

PILLAR provides a great flexibility for modeling the service. The service could be advertised in many different domains with different ontologies achieving different goals within different contexts and supporting multiple deployment models. PILLAR provides the basis for automatic service composition and automatic service recovery. PILLAR provides a good basis for defining service rating and building service repu-

tation models [12] for example services with less technical context requirements could be considered better than the services that require many technical context requirements.

7 Conclusion

This paper proposed the PILLAR model that captures the semantics of both the service and the user. PILLAR adopts the G^+ extended goal model. We proposed the G^+ model to represent the service/user goals, scenarios and contexts. PILLAR allows different ontologies to provide the service description with the basic vocabulary. We proposed an ontology meta-structure that defines the basic required ontology components for semantic discovery process. PILLAR provides the basic framework for automatic service semantic discovery and facilitates automatic service composition, automatic service recovery and service semantic clustering.

There is still some work to be done, such as devising an efficient semantic matching and semantic clustering techniques based on the PILLAR model. Matching SAVs with UAVs is not a simple operation, it requires different mediation levels and ontology mapping techniques. Clustering of SAVs is a key for achieving faster results. We need to define the similarity degrees between SAVs and devise the suitable distance function.

Acknowledgement. This paper is supported by the ARC(Australian Research Council) Linkage no LP0347217 and SUN Microsystems.

References

1. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng, "DAML-S: Semantic Markup for Web Services", Proc. of In International Semantic Web Working Symposium, 2001.
2. D. Chakraborty, F. Perich, S. Avancha, A. Joshi, "Dreggie: Semantic service discovery for m-commerce applications", Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems, 2001.
3. A. Dardenne, A. Lamsweerde, S. Fickas, "Goal-directed Requirements Acquisition", Science of Computer Programming, Vol 20, 1993, pp. 3–50.
4. D. Fensel, C. Bussler, "The Web service Modeling Framework WSMF", Electronic Commerce Research and Applications, Vol. 1, No. 2, 2002, pp. 113–137.
5. C. J. Fidge, "Contextual Matching of Software Library Components", Proc. Of the 9th Asia-Pacific Software Engineering Conference (APSEC), Australia, 2002, pp. 297–307.
6. S. Gribble, M. Welsh, R. Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services", IEEE Computer Networks, Special Issue on Pervasive Computing, Vol. 35, No. 4, 2001, pp. 213–230.
7. E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services", IEEE Internet Computing, Vol. 3, No.4, 1999, pp. 71–80.
8. HP Corporation, "E-speak Design and Architecture", <http://www.e-speak.net>, 1999.
9. W.K. Keith, "Core JINI", Upper Saddle River, NJ: Prentice Hall, 1999.

10. C. Lee and A. Helal, "A Multi-tier Ubiquitous Service Discovery Protocol for Mobile Clients," Submitted, to the First ACM International Conference on Mobile Systems and Applications (Mobisys), May 2003.
11. L. Liu, E. Yu, "Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach", Information Systems (journal), to appear
12. E.M. Maximilien, M. Singh, "Conceptual Model of Web Service Reputation", SIGMOD Record, ACM Special Interest Group on Management of Data, Vol. 31, No. 4, 2002, pp. 36–41.
13. R. McGrath, M. Mickunas, R. Campbell, "Semantic Discovery for Ubiquitous Computing", url = "citeseer.nj.nec.com/485904.html.
14. R. McGrath, "Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing", Department of Computer Science University of Illinois Urbana-Champaign http://www.cs.uiuc.edu/Dienst/UI/2.0/Describe/ncstrl.uiuc_cs/UIUCDCS-R-2000-2154.
15. S. McIlraith, C. Son, H. Zeng, "Semantic Web services", IEEE Intelligent Systems, Vol. 16, No. 2, 2001, pp. 46–53.
16. Microsoft Corporation, "Universal Plug and Play Device Architecture Reference Specification", <http://www.microsoft.com/hwdev/UpnP>, 1999.
17. OASIS consortium, UN/CEFACT, "ebxml", <http://www.ebxml.org>, 2001.
18. M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, "Semantic Matching of Web Services Capabilities", International Semantic Web Conference (ISWC), Sardinia, Italy, 2002, pp. 333–347.
19. C. Rolland, C. B. Achour, "Guiding the Construction of Textual Use Case Specifications", Data & Knowledge Engineering Journal Vol. 25, 1998, pp. 125–160.
20. C. Rolland, C. Souveyet, C. B. Achour, "Guiding Goal Modelling Using Scenarios", IEEE Transactions on Software Engineering, Special Issue on Scenario Management, Vol. 24, No.12, 1998, pp. 1055 –1071.
21. Salutation Consortium, "Salutation Architecture Specification", Version 2.1, <http://www.salutation.org>, 1999.
22. K. Sycara, S. Wido, M. Klusch, J. Lu: "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace." Autonomous Agents and Multi-Agent Systems, Vol. 5, 2002, pp. 173–203.
23. G. Wiederhold, "The Conceptual Basis for Mediation Services"; IEEE Expert, Intelligent Systems and their Applications, Vol.12, No. 5, 1997, pp. 38–47.

A Holonic Multi-agent System for Robust, Flexible, and Reliable Medical Diagnosis

Rainer Unland

Institute for Computer Science and Business Information Systems
University of Duisburg-Essen
45117 Essen, Germany
unlandR@cs.uni-essen.de
<http://www.cs.uni-essen.de/dawis/>

Abstract. Reliable, cost-efficient, and fast medical diagnosis, especially in crucial cases, is still a big challenge in today's world. This paper presents a holonic medical diagnosis system that combines the advantages of holonic systems and multi-agent systems in order to implement a highly efficient and robust Internet-based diagnosis system for diseases. The proposed holonic medical diagnosis system consists of a tree-like structured alliance of medical experts - realized by agents - that collaborate in order to provide a viable medical diagnosis. Each agent has a certain responsibility. Agents at higher levels of the holarchy represent experts in a broader field of diseases while the leaves are experts in one specific disease. The goal-based decision-making process is based on a hybrid, i.e., reactive and deliberative architecture. Reactive behavior is needed to compare the information provided to the agent with its pattern base in which possible appearances of the disease are stored in form of patterns. Deliberative behavior is needed in order to interpret and put in context the original data and information provided to the agent and to analyze and evaluate the results of the pattern matching process.

1 Introduction

In today's global world a fast and reliable medical diagnosis generation is of eminent importance as can be seen for example from the recent problems with SARS. Such highly contagious and lethal diseases can threaten the world if they are not fought immediately and with high efficiency and success. However, to do so it is, first of all, necessary to quickly and surely diagnose this disease if a person is suffering from it - and this regardless of where this person currently may stay in the world. While, after a short while, the identification of the disease at its hot spots may become routine its diagnosis at more remote/unlikely places will remain the real challenge.

In this context the paper discusses an architecture for a multi-agent system that relies on the holonic paradigm. The holonic entities (holons) are realized by agents that inherit many of the key properties of holonic systems, like self-sustainability, cooperation, self-organization, flexibility, robustness, and re-configurability. A holonic medical diagnosis system is a system of collaborative medical entities and experts that

work together to decide on a viable medical diagnosis in case of an uncertain situation (from the point of view of the responsible physician(s)). The diagnosis holarchy exposes a tree-like structure in which each inner node is an expert on a specific domain of diseases. Higher levels in the tree cover broader domains, however, on a more general level. Lower levels are more specialized, however, also cover a less comprehensive domain of diseases. Each leaf of the tree represents an expert on a specific disease.

Each agent of the holarchy has a certain responsibility. To fulfill its goal-based decision-making is based on a hybrid, i.e., reactive and deliberative architecture. Reactive behavior is needed to compare the information provided to the agent with its pattern base in which possible appearances of the disease are stored in form of patterns. Deliberative behavior is needed in order to process the original data and information provided to the agent and to analyze and interpret the results of the matching process. Representing holons as software agents enables the development of medical diagnosis systems as web-centric holarchies with an unlimited source for information access.

This paper is structured as follows. The next section discusses the state-of-the-art. Section 3 introduces basic concepts. Starting with holonic organizations it will expand this concept to holonic multi-agent systems. Section 4 will show how the concept of a holonic MAS can be adopted to implement a holonic medical diagnosis system (HMDS). It will be shown how the architecture of such a system looks like and how the general cooperation protocol for diagnosis generation works. Finally, section 5 concludes the paper.

2 Related Work

The area of medical diagnosis has long been intensively discussed in the area of artificial intelligence (AI), especially in expert systems (see, e.g., [17]). The probably best known expert system is MYCIN (cf. [18]). It is a program for advising physicians on treating bacterial infections of the blood and meningitis. It does reasonably well without common sense, provided the user has common sense and understands the limitations of the expert system. The problem with MYCIN, as well as with expert systems in general, is its flexibility, adaptability and extensibility. As long as rule bases are pretty small expert systems can be highly effective. However, the extension of rule-bases, e.g., as part of a learning process, can cause serious problems since it can easily happen that the overall semantics and behavior of the rule base gets out of control. Moreover, the extension to different or new areas of diagnosis is also not straightforward and easily exhibits the limitations of expert systems. Mainly for these reasons, expert systems had some success in specific, mainly quite narrow fields of expertise, however, always had problems to cover broader areas of expertise ([17]). MAS are much more flexible since each agent in such a system may only cover a very limited but concise area of knowledge. Complex and broad areas of expertise are covered by a cooperative effort of a number of agents. Thus, the overall system is highly flexible and adaptable and has the potential to extend its capabilities by gradual, controlled growth (adding of some new knowledge) by adding new units (agents) instead

of the necessity to extend a monolithic knowledge base. We do not want to discuss the different approaches that have evolved in the area of AI. Instead, we will concentrate on approaches that have emerged in the area of distributed AI, especially multi-agent systems.

Today, agent technology is on the brink to penetrate health related applications. However, given the high potential of this technology in this area it is surprising how few projects use it yet. Applications can mainly be found in the areas of patient monitoring, health care and patient management.

Patient Monitoring: The GUARDIAN system [1] manages care-related issues of patients in a Surgical Intensive Care Unit (SICU). The patient care model in a SICU is essentially that of a team, where a collection of (artificial) experts with distinct areas of expertise cooperate to organize the patients' health care. The artificial experts are represented by three different types of agents: *perception/action agents*, *reasoning agents* and *control agents*. They are organized in hierarchies and collaborate in order to lay down how patients can be treated in a preferably optimal way. The system relies on an adapted blackboard model for control and communication.

Patient Management: Managing patients ([2], [3]) in a shared-care context is an ambitious activity that requires a variety of profound knowledge. To support cooperative work in medical care, computer technologies enlarge the capabilities of individual specialists by enriching their ability of interacting with each other and with computational resources. The above papers propose a methodology for implementing a network of cooperating agents aiming to improve the health care delivery process.

Health Care: A prototypical agent-based distributed medical care system is described in [4]. Its main design issue is to unify and manage the patient management process. Such a process typically involves many activities and individuals. The system permits a natural representation of the care process, with agents representing individuals and, potentially, organizations involved in the patient care process.

Another paper ([5]) in this area introduces the concept of ubiquitous healthcare. Access to health services by individual consumers is possible via mobile computing devices. Addressing the related IT challenges - e.g., patient-centric knowledge processing, situation awareness, local control - the paper proposes the OnkoNet Mobile Agents Architecture. OnkoNet distinguishes between a macro and a micro level. It supports several cooperation protocols, inference models that control the system behavior, and a health ontology.

Ulieru et al. ([6], [7], [8]) propose the idea of internet-enabled soft computing holarchies for e-Health applications. Their approach builds on the synergetic triad Soft Computing - Internet - Multi Agent Systems in developing technologies for remote diagnosis, prediction and ubiquitous healthcare. This work is pioneering in the sense that it is, as far as we know, the first work that proposes an adoption of the holonic paradigm to the medical domain. Representing holons as software agents enables the development of e-Health environments as web-centric medical holarchies with a wide area of application in tele-medicine. The proposed approach exploits the triad's synergy twofold. On one side, it uses soft computing to enhance the power of the Internet by a fuzzy-evolutionary approach enabling emergence of virtual communities in Cyberspace. On the other side, it uses the Internet to empower soft computing both as enabler for the powerful integration of several soft computing strategies into a unified diagnosis and prediction methodology and as enhancer of learning

neuro-fuzzy diagnostic rules and knowledge base refinement through a remote database creation and exploitation mechanism. The work described in our paper sets up on this work, especially by adopting the idea of a holonic medical system.

3 Basic Concepts

3.1 The Holonic Paradigm

To design and implement systems that include both hierarchical organizational structures as well as decentralized control the Hungarian philosopher Arthur Koestler proposed a *holonic*-design. The underlying concept of such a design are *holons*. A holon describes a basic unit of an organization in social and biological systems (cf. [9]).

Holons may consist of sub-ordinate holons and, in turn, may be part of a larger whole. Thus, they form a tree-like hierarchy, called *holarchy*. Its members cooperate in order to achieve a complex goal or objective. Thus, a holarchy is often formed in an ad-hoc way and only exists temporarily until its overall goal is achieved. The holarchy defines the basic rules of cooperation between holons. When need arises, for example in order to safely and efficiently achieve a higher order goal, higher-level holons can deliberately reduce the autonomy of their subordinated lower-level holons.

A holarchy that is not part of an encompassing holarchy is a *holonic system / organization*. The strength of a holonic organization is that it enables the construction of complex systems that are nonetheless efficient in the use of resources, highly resilient to disturbances (both internal and external), and highly adaptable to changes in their environment.

The stability of holons and holarchies stems from the fact that holons are self-sustained units that have a degree of independence. They handle circumstances and problems on their particular level of existence without need of support by higher-level holons. On the other hand, holons may receive instructions from and may be controlled to a certain extent by higher-level holons. The self-contained characteristic of holons ensures that they are stable and capable to survive disturbances. The subordination to higher-level holons ensures the effective operation of the larger whole. On a more abstract level, three general layers of a holonic organization can be identified (cf. [10], [13]):

- ***The Inter-organizational Level.*** This highest level is formed by a number of independent organizations that (temporarily) join forces in order to address a complex higher level goal or to provide a complex higher level service that exceeds the specific capabilities of each individual organization.
- ***The Intra-organizational Level.*** Each member organization of the comprising holonic organization is capable to fulfill a specific (set of) service(s)/task(s). To do so it relies internally on a number of holons each of which recursively may consist of sub-holons.
- ***The atomic resource level.*** This level forms the leaf level of the holonic organization. It comprises those units of an organization that deliver basic tasks, i.e., tasks that can exclusively be performed by the corresponding holon. Thus, they can be

seen as atomic tasks and the corresponding holon as a specialist for the efficient and satisfactory execution of such a task.

3.2 Holonic Multi-agent Systems

Holonic-multi-agent-systems combine the concepts of holonic and multi-agent system (MAS) in order to create a holonic MAS. It consists of autonomous agents with *holonic* properties that recursively group together to form holons and, finally, holarchies (cf. [10], [11], [12], [13]).

In every non-leaf holon exactly one (sub-)agent acts as a facilitator ,i.e., represents the interface between the agents in the holon (*inbound duties*) and those outside the holon (*outbound duties*). This agent is the mediator agent of the holon.

The inbound duties of the mediator range from pure administrative tasks to the authority to issue directives to the subordinated agents of the holon. Thus, it may broker and/or supervise the interactions between the sub-holons of that holon. Furthermore, the mediator may have the authority to allocate resources to its subordinated agents or to remove parts from or assign new parts to the holon (cf. [13], [14]). The outbound interface of a mediator implements the interaction of the holon with the rest of the agent society. The mediator can plan and negotiate for the holon on the basis of the plans and goals of its sub-agents.

4 Architecture of the MAS-Based Holonic Medical Diagnosis System (HMDS)

Medical diagnosis serves the purpose to identify the disease a patient is suffering from and, then, to determine how it can be treated best. Such a process usually relies on all kind of new and old data. New data is gained by interrogating the patient and by conducting necessary (tool-driven) examinations and tests. New and old medical data and information about patients is usually stored in medical records for patients or patient records for short. A patient record usually contains information about prior diseases of the patient, possible allergies, conducted medical surgeries, therapies that have been performed, diagnoses that were made, (current) symptoms and much more. Unfortunately, today still a number of different patient records are maintained at different places on different levels of abstraction. These records are usually not related. Instead, they are kept as “private” property of the physician or healthcare institution that originally initiated its collection. Thus, this information and data is not accessible for other medical institutions or physicians.

In this paper, we will assume that all necessary/available medical information about a patient is kept in one unique and comprehensive patient record. Such a human readable document needs first to be translated into a computer readable document in order to be processible by a computer. This paper will not discuss this translation process. Instead, it is assumed that such a computer readable form is already available. It will be called *computer readable patient pattern* (CRPP). We call it pattern since in our system it will be used to compare it to patterns that describe specific diseases; i.e.,

our approach relies on pattern matching. In order to be computable by agents a CRPP needs to be associated with an appropriate ontology. This ontology defines how the agent may understand, interpret and process the CRPP. Again, we will not deal with this topic in this paper. Instead, it is assumed that a CRPP comes with an appropriate ontology that provides all information necessary for agents to understand and further process the CRPP.

Our system supports medical institutions in generating appropriate diagnoses and treatments in difficult medical cases. A medical institution can be a clinic/hospital or an individual physician. Difficult cases are those in which the patient's symptoms do not sufficiently match typical disease patterns known by the medical institution. Such uncertainties may occur if

1. a patient suffers from a combination of diseases.
2. a patient does not exhibit the typical symptoms of a specific disease, even so he suffers from it.
3. the disease is not yet known to the medical institution because it is either new or usually only appears in specific regions of the world that are far away from the place where the patient is treated right now.

The last case usually means that the symptoms may or can not be interpreted properly.

In the following a holonic medical diagnosis system will be proposed that relies on a multi-agent system architecture as well as on the holonic paradigm. In allusion to the concept of holonic manufacturing systems (HMS) it is called *Holonic Medical Diagnosis System* (HMDS).

In accordance to the holonic paradigm HMDS also exhibits the three fundamental layers.

The most important layer is the intra-organizational one. It consists of a set of holons/ agents that are arranged in a tree-like structure (see Fig. 1). The interface of this layer to the outside is realized by a special agent, the mediator agent. It represents the unique entrance to the HMDS; i.e., all communication to and from the outside world is done via this mediator agent.

HMDS supports a medical institution if it is not confident enough in its diagnosis. In such a case an appropriate (excerpt of the) CRPP is produced and send to the HMDS. Inner holons of the HMDS that represent so called *diagnosis specialist agents* (DSAs) take the CRPP and decide to what other agents they are to be sent for further processing.

So-called *disease representative agents* (DRAs) represent the basic resource or leaf level of the HMDS. Each DRA is a representative of a specific disease. If it is assigned a diagnosis request, it takes the CRPP, analyses it and computes a result.

From the above-mentioned three reasons for uncertainties in a reliable diagnosis, the first two can probably be dealt with directly by the HMDS. However, the last reason may require external expertise since appropriate knowledge may not be available within the HMDS. In such cases, the highest layer, the external layer, comes into play. A request for external advice is sent to the mediator agent of the HMDS. It disseminates it to appropriate other HMDSs or adequate information sources. It may even be appropriate that a number of external entities (holonic organizations) join forces to collaboratively deal with the problem, e.g., in case the disease is so far unknown.

4.1 Intra-organizational Level: Diagnosis Specialist Agent (DSA)

The intra-organizational level of the HMDS consists of a tree like set of diagnosis specialist agents (see Fig. 1). Each DSA represents a specialist in a specific domain of related diseases. If it gets a request for a diagnosis, it interprets the CRPP in order to decide to what other DSA(s) or DRAs the CRPP is to be sent for further processing. Furthermore, after the DRAs have returned their diagnosis it interprets, compares, and evaluates them in order to define a ranking of likely diagnoses among all the diagnoses returned from different DRAs or DSAs. Diagnoses may be on different levels of certainty. In the best case, the information that was provided to the DRA, especially the CRPP, provides all the data and information that is needed in order to conduct a comprehensive search on the patterns of its pattern base. In such a case the results achieved by the DRA are optimal from its point of view and on the basis of its current status of knowledge, i.e., cannot be improved by any further information or data.

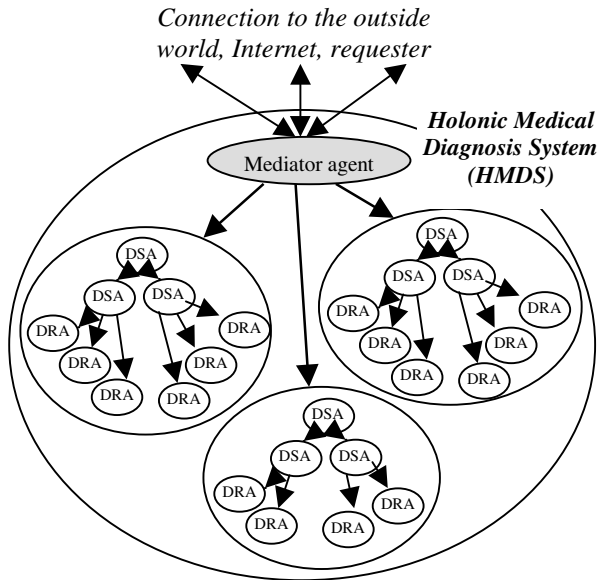


Fig. 1. Model of a Holonic Medical Diagnosis System (adapted from [6])

In less fortunate cases the CRPP may only provide a (substantial) part of all information and data for an optimal diagnosis. This influences the significance of the prediction of the appropriate DRA. Thus, in case the already available information in the CRPP does not exclude this disease the result of its analysis comes with a set of tasks, examinations, and tests that are suggested to be performed by the medical institution in order to further verify (or invalidate) the diagnosis. This clearly requires a deliberative agent architecture with the DSA exhibiting a profound and comprehensive understanding of the domain of disease it represents.

DSAs can be arranged in a hierarchical way, which means that, on a higher level, a DSA is a representative of a superset of the set of diseases that is represented by the

next lower DSAs. Moreover, alternative DSAs on the same level may represent different doctrines or different views of a disease expert. For example, one DSA may represent the doctrine of traditional medicine while another one represents the homeopathic doctrine.

An important task of a DSA is to interpret the incoming diagnoses and come to a conclusion on whether there is sufficient evidence for the likelihood of a specific disease. If not, the DSA has to decide - on the basis of the delivered diagnoses/information - whether it makes sense to consult other DRAs or, if this seems to be unpromising, whether it seems to be more promising to advertise the request on the Internet. This is especially promising if there is a suspicion that the disease is a so far unknown or imported one, thus one that is very rare in the living space of the patient/medical unit.

4.2 Basic Resource Level: Disease Representative Agent (DRA)

Each DRA is a representative and an expert on a specific disease. It maintains a pattern store or database that contains disease description patterns (DDP) for all possible/known variants/appearances of the disease. Therefore, on a first glance, such an agent can be seen as a reactive agent that reacts on a pattern match basis to CRPPs that are sent to it. However, it is unrealistic to assume that the CRPP and the DDP match in their structure and semantics. Thus, the DRA needs to be enriched with sufficient knowledge to be able to interpret the CRPP and to relate it to its DDPs. Moreover, it may be that relevant data/examinations are missing what may compromise the significance of other characteristics of the CRPP. Therefore, the DRA needs to be equipped with deliberative capabilities as well.

The key component of a deliberative agent is a central reasoning system ([15]) that constitutes the intelligence of the agent. Deliberative agents generate plans to accomplish their goals. A (restricted and focused) world model may be used in a deliberative agent, increasing the agent's ability to generate a plan that is successful in achieving its goals. In accordance to these requirements, every DRA possesses an internal representation of the disease it represents (a "disease model") that allows it to interpret the CRPP and compare it to its DDPs. After the comparison, its results are interpreted and a probability value for the occurrence of the disease as well as an estimation of the reliability of the diagnosis (acatalepsia) are calculated. Moreover, the agent enhances its result with suggestions on how it can be improved/verified if not all relevant information were available to the agent. If, for example, necessary examinations were not conducted yet or highly relevant characteristics were not considered/collected yet this hint is added to the diagnosis results of the agent. Additionally, in case the likelihood of the disease is above a given threshold proposals for the treatment and information about the disease are added.

To summarize, a DRA relies on a hybrid architecture. It comes with a set of patterns that describe all known appearances of the disease. Moreover, it comes with an extensive set of rules that characterize the disease and places it in context. This requires that the DRA is equipped with an extended disease model that allows it to interpret and interrelate the characteristics of the CRPP and to draw reliable conclusions with respect to the likelihood of the occurrence of the disease.

From a logical point of view, the structure of the HMDS may not form a tree but a graph. This is because there may be overlaps between different areas of disease, thus, between DSAs and DRAs that represent them. However, in our implementation we clone agents as often as they are needed. Thus, on the physical level the structure forms a tree. We have integrated an evolutionary learning strategy to tune the agents of the HMDS. Based on experiences and feedback the rule set of each agent is constantly monitored, evaluated on its success and, in dependence of the evaluation, improved and fine-tuned¹. This means that we have to deal with the problem of duplication. In case of the installation of an improved knowledge base all agent instances need to be updated that represent a (cloned) instance of the underlying agent type. Such information about duplicate agents is stored and maintained and will be used to guarantee a conflict-free update of agent instances. On the other hand, a specific agent type that is added to different holons may develop differently. This means that these agents are only the same in the beginning, however, are not equal. They may develop in different directions by learning differently. For example, a DRA that represents a specific disease under two different DSAs (e.g. that represent different doctrines) may be more successful in its prediction under the one doctrine than under the other. Thus, as part of the learning process both instances will improve in different directions.

4.3 Inter-organizational or Global Collaboration Level

All physically coherent units, e.g., hospitals, medical clinics or research institutes, that can be addressed in the outside world and that can contribute to the diagnosis generation process, form the inter-organizational level. We assume that each such unit is represented by a mediator agent that serves as the unique entrance to the interior of this unit. The mediator decides whether it accepts a request from the outside for help on diagnosis generation. The decision may depend on the assessment of the likelihood of the organization to be able to come to a solid diagnosis or simply on the fact whether there are enough resources available to do the job.

In order to be able to quickly address the right units for a request for help in the diagnosis generation process it is helpful if some broker (facilitator) or matchmaker agent exists that is capable to choose the most relevant candidates on the basis of the provided CRPP.

A broker agent actively intermediates requested services by taking charge of all communication between requester and provider agents. It typically contacts (a set of) the most relevant provider agent(s), negotiates for, executes and controls appropriate transactions, and returns the result of the services to the requester agent.

The task of a matchmaker agent is much simpler. It simply pairs requester with provider agents by means of matching given requests with appropriate advertised services of registered/known provider agents. The matchmaker returns to the requester a ranked list of advertisements and the contact information of the agents whose advertisement matches the request. The requester chooses the most suitable agent for its needs and interacts/negotiates directly with it to get the requested service.

¹ We are currently working on a learning component for HMDS based on neural networks. Although we already have some concrete ideas it is too early to present them here. We are aware that this learning component is crucial for the HMDS.

4.4 The Cooperation Protocol for Diagnosis Finding

Fig. 2 describes how the diagnosis generation process works. The requester, usually a medical institution, sends its request for a diagnosis together with the corresponding CRPP to the mediator agent of the HMDS. Depending on the kind of request, the mediator decides whether the request is propagated to one or many DSAs. This decision depends on the kind of request of the medical institution. If diagnoses from different doctrine agents are requested then the request will be send to the specified ones. This implies that each doctrine agent will send its result/diagnosis to the requester (independently of the diagnoses of the other doctrine agents).

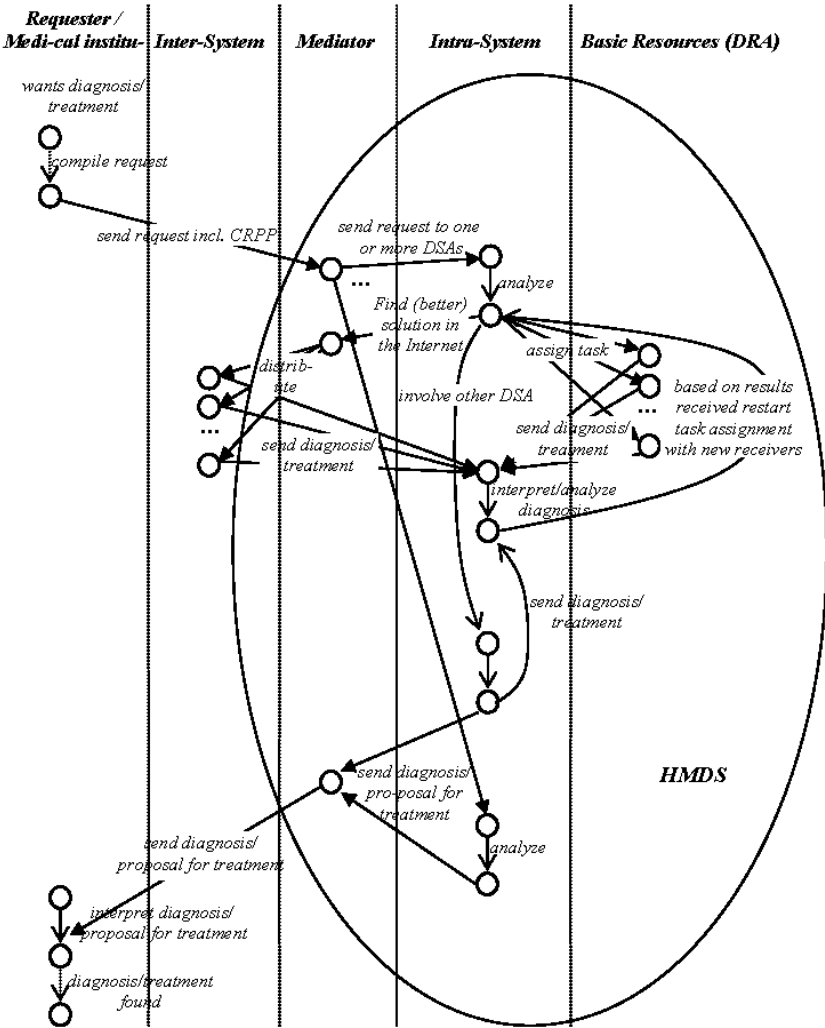


Fig. 2. Cooperation protocol for diagnosis generation

After the mediator has distributed the request to the appropriate “top-level” DSAs each DSA analyzes the request and the CRPP in order to find out what lower level DSAs or DRAs are to be assigned the request. Depending on the internal structure of the intra-organizational level the request may recursively migrate from a more general DSA to one or more specialized DSAs. Each DSA on the path down has a more profound knowledge in a more specialized area of diseases. Finally, the request is assigned to one or more DRAs. The DRAs analyze, interpret and evaluate the CRPP of the request and match the relevant information of it with its own pattern base. All patterns that exceed a given threshold are further analyzed in order to identify the most promising one and to calculate the probability of the disease and the reliability of the diagnosis. This information together with possible treatments and further suggestions for examinations and tests are sent back to the next higher DSA. The DSA gathers the results from all DRAs it had assigned the diagnosis request and evaluates and compares them. If the DSA – based on the achieved results – comes to the conclusion that the original assumption with respect to the possible disease may no longer be valid it may decide to start the diagnosis generation circle again, this time by assigning the task to other agents. This implies that the DSA may conclude that none of its subordinate agents can come to a reliable diagnosis, hence that the superordinated DSA has chosen an inappropriate DSA or there exists no appropriate DSA in the HMDS. In the latter case, the request needs to be sent to the inter-organizational level; i.e., will be announced/advertised on the Internet. Such a decision can only be made by a top-level DSA. Each intermediate DSA may report a suggestion for a recomputed task assignment together with corresponding evidence and relevant data to the higher level DSA.

Finally, when a high-level DSA gets a diagnosis that, from its point of view, is solid, stable, and (close to) optimal with respect to the given situation and knowledge of the HMDS it sends it together with all other information via the mediator agent to the requester. If it, after the analysis of the results, comes to the conclusion that it can be improved or is altogether not satisfactory enough it may decide to either involve other DSAs (DRAs) in the diagnosis process or to consult the outside world.

4.5 Middleware and Agent Platforms

Currently, we are working on the implementation of the intra- and basic resource level of HMDS. We have not invested much work in the inter-organizational level. This is due to the high complexity of this level and the many activities that are to be performed in order to make this level a valuable one. In a first step, we will develop several HMDS and allow them to cooperate. This will form a restricted version of the inter-organizational level, however, will allow us to gather a whole bunch of necessary experiences in order to address this layer on a broader level later on.

An HMDS can be realized with the help of an agent platform (like, e.g., JADE, [19], [20]). JADE is a middleware for FIPA²-compliant multi agent systems. It supports application agents whenever they need to use features covered by the FIPA standard specification (agent life-cycle management, message passing, etc.). The

² Foundation of Intelligent Physical Agents; standardizes agent technology; <http://www.fipa.org>

communication infrastructure of JADE offers flexible and efficient messaging, transparently choosing the best transport available. Although appearing as a single unit to the outside world, a JADE agent platform is a distributed system. It can be split over several hosts with one of them acting as a front end where Agent Management System and yellow page service agents are placed. A JADE system comprises one or more Agent Containers, each living in a separate Java Virtual Machine and delivering runtime environment support to some JADE agents.

An agent platform needs to contain a component called Agent Communication Channel, whose task is to transparently provide a Message Transport Service, relying upon at least one FIPA compliant Message Transport Protocols (MTPs). FIPA has defined MTPs for IIOP, HTTP and WAP protocols, so interoperability can be achieved exploiting any among the three protocols above. It can be distinguished between inter-platform messaging (the sender and the receiver agents live on different platforms (belong to different HMDS)) and intra-platform messaging (the two interacting agents are within the same platform (belong to the same HMDS)). While inter-platform messaging has to comply with FIPA specifications, intra-platform message delivery can be realized according to the specific needs of the underlying agent platform. Relying of such agent platforms as basis for the construction of an HMDS permits to abstract from the specific needs and demands of the underlying middleware. Thus, we can concentrate on the architecture and implementation of the HMDSs.

5 Conclusion

In today's ever-changing world that grows together ever faster, a quick, ubiquitous, and reliable diagnosis generation is of eminent importance as can, for example, be seen from the recent problems with SARS. Also, the problems with the cost explosion in the health care sector ask for a highly efficient, cost-effective and reliable health care system. In this paper we have presented a holonic medical diagnosis system that unifies the advantages of multi-agent system technology with the efficiency, reliability, extensibility, and flexibility of the holonic paradigm. The resulting system supports the medical institution in its diagnosis generation, especially in cases in which it has only vague assumptions about the kind of disease the patient is suffering from. The kernel of our system, the HMDS, consists of a tree-like hierarchy of agents with each internal agent representing a specialist in a more or less broad domain of diseases (depending on its position in the tree) while the leaves exhibit profound knowledge on one specific disease. Coordinated by the higher-level agents the system successively translates the diagnosis request in self-contained subtasks for further processing by specific agents. For their diagnosis generation they are provided with the necessary data/information. After the expert agents have performed their job (generated a diagnosis) they deliver their results to their immediate higher-level agent. Higher level agents gather the results from all those immediate lower-level agents to which they had assigned a diagnosis request. They analyze, relate, and evaluate these results in order to come to a more comprehensive and reliable conclusion. In case the results of such an analysis are not satisfying the agent may decide to announce the problem (request for a diagnosis) to the outside world, which means that other in-

stances - spread over the Internet - that may be able to contribute to the diagnosis generation will be contacted for help.

References

- [1] Hayes-Roth, B.; M. Hewett, R. Washington, R. Hewett, and A. Seiver. *Distributing intelligence within an individual*. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989, pp. 385–412.
- [2] Lanzola, G.; S. Falasconi, and M. Stefanelli. *Cooperative Agents Implementing Distributed Patient Management*. In: Proceedings of the MAAMAW-96 Conference. Berlin: Springer, 1996
- [3] Lanzola, G.; S. Falasconi, and M. Stefanelli. *Cooperative Software Agents for Patient Management*. In: Proceedings of the AIME-95 Conference, pp. 173–184; 1995
- [4] Huang, J.; N.R. Jennings, and J.Fox. *An agent-based approach to health care management*. Int. Journal of Applied Artificial Intelligence 9(4): 401–420, 1995.
- [5] Kirn, St.: *Ubiquitous Healthcare: The OnkoNet Mobile Agents Architecture*; In: Objects, Components, Architectures, Services, and Applications for a NetworkedWorld;. Aksit, M.; Mezini M.; Unland, R. (Eds.); Invited and revised contributions of 3.rd Int. Conf. Netobjectdays (NODE) 2002, Erfurt, Germany, October 7–10, 2002; Lecture Notes in Computer Science (LNCS) 2591; Springer Publishing Company; March 2003
- [6] Ulieru, M.: *Internet-Enabled Soft Computing Holarchies for e-Health Applications*, Invited Chapter in: Fuzzy Logic and the Internet; Zadeh, L.A.; M. Nikravesh (Eds.), Physica Verlag; 2003
- [7] Ulieru, Mihaela and Geras, Adam: *Emergent Holarchies for e-Health Applications: A Case in Glaucoma Diagnosis*; Proc. IECON '02; HOL - Agent-based Intelligent Automation and Holonic Control Systems; Sevilla, Nov. 2002
- [8] Ulieru, M.; O. Cuzzani, S. H. Rubin, M. G. Ceruti: *Application of Soft Computing Methods to the Diagnosis and Prediction of Glaucoma*, Proc. of the 2000 IEEE International Conference on System, Man, and Cybernetics, SMC 2000, “Cybernetics Evolving to Systems, Humans, Organizations, and their Complex Interactions”, Nashville, TN, USA, Oct. 8-11. 2000,
- [9] Koestler, A: *The Ghost In The Machine*, Arkana Press.; 1967
- [10] Ulieru, M, Brennan, R, Walker, S.: *The Holonic Organization – A Model for Internet-Enabled Global Supply Chain and Workflow Management*. In: International Journal of Integrated Manufacturing Systems, No 13/8, ISSN 0957-6061; 2002
- [11] Fischer, K.; Funk, P.; Ruß, C.: *Specialized Agent Applications*; in [16]; 2002
- [12] Deen, S. M. (Editor): *Agents Based Manufacturing: Advances in the Holonic Approach*; Springer Verlag; ISBN 3-540-44069-0; 2003
- [13] Ulieru, M: *Emergence of Holonic Organizations from Multi-Agent Systems: A Fuzzy-Evolutionary Approach*, Invited Chapter in: Loia V (Ed) *Soft Computing Agents*, IOS Press; 2002
- [14] Maturana, F; Norrie, D.: *Multi-Agent Mediator Architecture for Distributed Manufacturing*. In: Journal of Intelligent Manufacturing, Vol 7, pp. 257–270. 1996
- [15] Ginsberg, M.: *Universal planning: An (almost) universally bad idea*. AI Magazine 10(4): 40–44. 1989

- [16] Luck, M.; Marík, V.; Stepánková, O.; Trappl, R. (Editors): *Proc. Advanced Course on Artificial Intelligence (ACAI 2001) "Multi-Agent Systems and Their Applications (MASA)"*, Prague, Czech Republic; July 2001; Joint event of ECCAI (European Coordinating Committee for Artificial Intelligence) and AgentLink (European Network of Excellence for Agent-Based Computing); Springer Lecture Notes in Artificial Intelligence (LNAI) 2086; 2002
- [17] Pople, H. E., Jr. *Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics*. Chapter 5 in Szolovits, P. (Ed.) *Artificial Intelligence in Medicine*. Westview Press, Boulder, Colorado. 1982
- [18] Shortliffe, E. H. *Computer-Based Medical Consultations: MYCIN*. Elsevier, New York, 1976
- [19] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa: *JADE – A FIPA-compliant agent framework*; Proceedings of PAAM'99, London, April 1999, p. 97–108.
- [20] Federico Bergenti, Agostino Poggi, Giovanni Rimassa, Paola Turci, Fabio Bellifemine: *Middleware and Programming Support for Agent Systems*; Dipartimento Ingegneria dell'Informazione, Università degli Studi di Parma, Parco Area delle Scienze, Parma, Italy

Choosing Reputable Resources in Atomistic Peer-to-Peer Environments

Georgios Pitsilis¹

University of Newcastle upon Tyne, U.K.,
School of Computing Science
Georgios.Pitsilis@ncl.ac.uk

Abstract. Peer-to-Peer information sharing environments have gained recognition and popularity during the recent years. In spite of the useful characteristics they provide in the ways that the participants can collaborate, the issue of quality preservation in the shared material has not been addressed yet. The lack of appropriate mechanisms and policies to evaluate the participants has sown fears that the overall popularity of the services will be affected. The nature of atomistic p2p models, where survivability is based on the idea of self-organization into communities could be the basis of a solution to the quality problem build-up by the peers themselves. We consider that the deployment of an assessment scheme as a consultancy service based on a localized view of reputation could help the associated members of the peer-to-peer community in making their choices and thus in the provision of better services.

1 Introduction

The term peer-to-peer has always been used to describe a connectivity architecture which in simple terms means equal communicating with equal. In such a network all the participants offer the same services and obey the same rules. The success of content sharing applications like Gnutella and Free-net has shown that this type of decentralization is likely to have a prosperous future.

2 The Problem

The issue of preserving QoS on services of the “static internet” is the only area that has been studied so far [1]. The global form of the contemporary peer-to-peer type of collaboration where every member within the infosphere is freely allowed to share any kind of material raises the issue of how the quality of service can be preserved in such a model of networking with no central consultancy service. The difference and also the difficulty in p2p systems is found in the dynamic way the peer communities are structured and especially in the *Atomistic* model where the peers cannot exist alone without mutually supporting each other. Unlike the static web, the provision of

¹ Scholar of Greek State Scholarships Foundation (IKY)

low quality services has a large impact on performance and can even lead to the collapse of the system [2] when the users receive dissatisfactory services.

The problem comes from the nature of the peer-to-peer model where virtual networks are built upon the collaboration of computers around the world. In this form of networking the nodes/peers are running mostly on ordinary workstations whose connectivity behavior is in fact unstable and unpredictable. This is either due to the use of unreliable transport media or it depends on the working habits of the users. These impact each workstation's ability to serve requests. For example a node that runs on a mobile host is less capable of serving a request due to the higher probability of going off-line during a retrieval operation and thus disrupting the service.

It is common practice on the static web today for similar situations to be addressed by the use of a centralized consultancy services like UDDI [3], or in the simplest way by meta-search engines, which attach as metadata semantic information to search responses. Even though these mechanisms are widely used, they also carry the weaknesses of centralization, (single point of failure, bottlenecks), and hence are unsuitable for P2P environments. The weakness such solution carry is that of showing bias in their judgments towards a particular option. This suggests the use of a decentralized solution.

Reputation based schemes in peer-to-peer networks have also been proposed in the past [4]. The proposed techniques are mainly based on polling mechanisms, which perform reliability checks on candidate participants prior to downloading by sending messages to all neighboring peers. Even though there are no evaluation results for the level of improvement this algorithm provides, the polling mechanism itself is in fact a centralized idea of communicating and obviously impacts the scalability of the system.

3 Our Proposed Solution

Our concept is to provide a solution based on techniques similar to those that are used in everyday life. This is based on the observation of the many similarities between p2p networks and social communities, especially in the ways they grow and exchange information, and the trust relationships build between their entities. In particular, whenever choices have to be made, people seriously consider factors such as *Reputation* and *Trust*. That happens because people tend to trust reputable services and their friends when they ask for opinions rather than other total strangers. Even though reputation theory is quite old as a research issue, it has been applied in computer science for the first time only recently. Reputation in general means "what we expect about an entity's behavior based on observations and collected information from the environment regarding the past actions of the entity"[5]. We believe that applying a reputation scheme in p2p similar to the one that has been developed within the human communities may benefit system performance by improving the success of queries and at the same time avoiding the problems of centralization.

Our proposed solution is to employ the "word of mouth" techniques to help each peer in the network to build and maintain reputation profiles of how they perceive other peers and to share this information for the benefit of the whole community. "Word of mouth" is a known recommendation mechanism that people use for infor-

mation diffusion and is basically the way that searching is carried out throughout social relationships. The idea is, people who have lived through an experience before can be a good source of information to others who might want to live the same experience in the future.

Newer organization schemes suggest that peers should get organized into groups based on their interests [6] in the same way that people rely on others who have similar interests. Even though this kind of organization has already been adopted in the newer p2p implementations [7], the information that is exchanged between the nodes (e.g. historical data of past behavior) is not used for the purpose of building reputation profiles.

In order to automate the above mechanisms of building social relationships, we consider the virtual community as a directed graph with weighted edges that correspond to the reputation ratings that one node gives to another. The graph grows as the experiences between the nodes increase and more reputation ratings take place. Under this scheme, each node, before choosing a peer, could gather via "word of mouth" enquiries as much information as possible, to build the reputation profile of the target peer. A typical reputation enquiry could be initiated via an ordinary broadcast, hopping from node to node recursively until it reaches the nodes close to the examined one. Any direct links with the target could denote the existence of personal experiences and thus their ability to providing personalized reputation ratings. The neighbors in the scheme act all together as a formed group and actually carry out this "survey" by providing opinions. It is worth pointing out that not all neighbors' opinions are given the same weight during the assessment, but the weight value is given in proportion to the reputation measure that corresponds to the "path" where the information comes from. The notion of intermediate reputation ratings offers a quite strong characteristic, as the actual reputation value can be determined considering all the intermediate nodes that carry the responses back to the peer where the query was originated. Likewise, in the real world example, the opinions are also freely formed. To develop a human like judgment for the model, we could re-rate the neighbor's reputation values, after a choice has been made, by giving new weights depending on how close the suggested reputation had been to the actual one.

In contrast to the traditional centralized reputation systems that require the existence of a central trusted entity as a deposit for the behavioral data (e.g. UDDI), our algorithm requires no central entities at all to store the reputations. Instead, that information is gathered dynamically whenever reputation queries are fired inside the system graph and the trust relations with other peers are built as the originating peer evaluates the reputation ratings.

The model we described for building reputation profiles employs operations analogous to the Histos [8] system, which was designed especially as a collaborative reputation mechanism for electronic marketplaces. That system was inspired by the *Friend of a Friend Finder* Scheme [9] where it is thought that any persons within a community can be known to each other through their relationships with their friends.. Respectively, any relationships established between entities could be based on the commonalities that can be found on their sets of interests.

We believe that this algorithm, if applied to P2P systems, will show powerful characteristics, such as the ability to generate reputation dependent on the point of view, which is a fundamental property of P2P. Nevertheless, P2P networks use for their connectivity a virtual network that links the nodes together, and not considering this along with the reputation norm would be a serious omission. Given that a P2P net-

work is substantially an application layer network built upon the Internet infrastructure, in our case, is desirable to know the physical network distances between the peers because the actual service provision is done at this level. Thus, a decision based on a hop distance is not always correct and it can lead to wrong estimates of a peer's "reputability". This is because the virtual network topology, as various studies about the Gnutella network infrastructure [10] have shown, mismatches the Internet topology where Gnutella runs. Changing the topology as a remedy is not always applicable and is prohibitively expensive. Instead, we could focus on the ways that gathered information is assessed and to consider the topology information to some extent in rating the reputability of a peer. That way, in our proposed scheme, the topology information is upgraded to an influential factor offering also personalized characteristics in the shaping of reputation.

4 Conclusion

In conclusion, we believe that the way that Reputation is built in human social communities provides a good analogy to studying the problems of quality of services in Atomistic peer-to-peer networks. The numerous benefits of distributed computing such as congestion avoidance and the non-existence of a single point of failure, that are met in this model, combined with the characteristics that are found on the human trading societies, are promising enough to build a useful platform. Moreover, considering that the underlying network infrastructure offers a strong dependency on the user's point of view, that is otherwise missing, it can provide enhanced characteristics and get the model closer to a real situation.

References

1. Raju Pandey - J. Fritz Barnes, "Supporting QoS in HTTP servers", Proceedings of the Seventeenth Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC). (Puerto Vallarta, Mexico, June 1998).
2. H. Chen, A.M. Schroeder, "A Modified Depensation Model of Peer to Peer Networks: Systematic Catastrophes and other potential Weaknesses", AMATH 383 - June 2002
3. Universal Description Discovery and Integration Technical White Paper. September 2000
4. E. Damiani - S. De Capitani -S. Paraboschi, P. Samarati, "A Reputation-Based approach for finding Reliable Resources in Peer-to-Peer Networks", CCS'02 Nov 18-22, 2002 Washington DC, USA
5. A. Abdul-Rahman - S. Heiles, "Supporting trust in Virtual Communities", proceedings of International conference On System Sciences, Jan 4-7 2000, Hawaii
6. K. Sripanidkulchai, B. Maggs, H. Zhang, "Efficient Content Location Using Interest Based Locality in Peer-to-Peer Systems", Infocom 2003.
7. M. Prinkey, "An efficient scheme for query Processing on Peer-to-Peer Networks", Aeolus Research Inc.
8. G. Zacharia - A. Moukas - P. Maes, "Collaborative Reputation Mechanisms in Electronic Marketplaces", Proceedings 32th Hawaii International conference on system Sciences - 1999

9. N. Minar- A. Moukas, "Friend of a friend finder", Software agents Group, MIT Media Lab, working paper 1988.
10. Matei Repeanu, "Peer to Peer Architecture case study: Gnutella Network", University of Chicago, Dept. of Computer Science, Tech Report TR-2001-26

A Fast Group Communication Mechanism for Large Scale Distributed Objects¹

Hojjat Jafarpour and Nasser Yazdani

Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
hjafarpour@ece.ut.ac.ir, yazdani@ut.ac.ir

Abstract. Group communication undoubtedly is a useful paradigm for many distributed object systems. Provision of a fast and scalable group communication mechanism can considerably improve the efficiency of these systems. In this paper, we propose a new group communication mechanism for large scale distributed objects over a huge heterogeneous inter-network. The proposed mechanism uses Message Dispatcher Objects, objects co-located with each group member to distribute and parallelize the group communication load among all group members. Our proposed mechanism also recovers from the host failure in the system. Experimental results show that our scheme scales well in a large number of group members.

1 Introduction

Group or multicast communication is an important communication model in distributed systems. A group can be a set of processors or objects. Group communication service's responsibility is to deliver messages to all the group members [2]. Indeed, they are powerful building blocks that facilitate the development of many distributed applications [1].

One of the main applications of group communication services is to provide fault tolerant capabilities for distributed object systems such as CORBA. These systems achieve fault tolerance and highly availability by means of object replication. They use group communication services to maintain consistency of replicated objects. For instance, OMG's Fault-Tolerant CORBA specification [3] recommends using group communication services to support active object replication in CORBA.

Group communication mechanisms are used in applications such as distributed transactions and database replication [4], highly available services [5], support for distributed and clustered operating systems [6] and collaborative computing [7].

¹ This work is supported as a research project by the Faculty of Engineering, University of Tehran, Tehran, Iran

Many group communication mechanisms for distributed systems have been proposed among them to mention Totem [8], Hourus [9], Transis [10] and Spread [11].

Among the main requirements of these systems are scalability of the group communication mechanism and low message delivery time. These features can considerably improve the performance and efficiency of applications using them. Unfortunately, majority of previously proposed mechanisms do not scale well with respect to the number of the group members on a huge heterogeneous inter-network like the Internet.

In this paper, we propose a fast and scalable group communication mechanism for distributed objects on a heterogeneous inter-network using application level communication techniques such as remote method invocation. The proposed approach achieves scalability and high message delivery speed by distributing and parallelizing the multicasting load among hosts in the system. It uses Message Dispatcher Objects (MDOs), which are objects co-located with each group member, to route and deliver multicast messages to the group members. A message is sent to a MDO which is the proxy of the group and is disseminated among MDOs using an efficient algorithm. Then, each MDO delivers the message to its co-located group members. The proposed mechanism also detects host failure and recovers from it.

The rest of the paper is structured as follow. Section 2 describes the proposed group communication mechanism. Section 3 presents how the mechanism recovers from host failure. In section 4, some characteristics of the mechanism and experimental results are presented. Section 5 reveals some directions for future work and concludes the paper.

2 The Proposed Group Communication Mechanism

2.1 System Model

The proposed mechanism assumes that group members are on different hosts in a huge heterogeneous inter-network such as the Internet and there is a logical path between every two hosts. The communication channels between every two hosts are reliable and communications are done using application level techniques such as remote method invocations.

Message Dispatcher Objects (MDOs): The main components of the mechanism are special objects co-located with group members which we call them MDOs. The mechanism supposes there is one MDO on each host in the system. Each MDO knows all MDOs and their addresses. This information is stored in a list called *MDO list*. The index of each MDO is the same in all MDO lists in the system. Figure 1 depicts a MDO list for a group with 16 members on 16 hosts. In such a system, there is one MDO object on each host and all of them have the depicted list. The first MDO in the list is called the *proxy MDO*. Each MDO also has a message storage queue that incoming messages are stored in. Using this queue, MDOs store a message for a limited period of time to ensure that all group members have received the message. If there are more than one group member on a host, its MDO knows all of them too.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MDO ID	MDO 0	MDO 1	MDO 2	MDO 3	MDO 4	MDO 5	MDO 6	MDO 7	MDO 8	MDO 9	MDO 10	MDO 11	MDO 12	MDO 13	MDO 14	MDO 15
Address	Host 0	Host 1	Host 2	Host 3	Host 4	Host 5	Host 6	Host 7	Host 8	Host 9	Host 10	Host 11	Host 12	Host 13	Host 14	Host 15

Fig. 1. MDO list which is the same for all MDOs in a system with 16 group members on 16 hosts.

2.2 Message Propagation Mechanism

When a group member wants to send a message to the group it does not send the message to the members directly. Instead, it sends the message to one of the MDOs which is called the proxy MDO. Then, MDOs propagate the message among themselves. In the next step each MDO delivers the message to its local group member(s). Message dissemination among MDOs is done in a parallel manner and the propagation load is distributed among hosts in the system.

The message propagation process among MDOs is as follow. Suppose the proxy MDO receives a message to propagate among MDOs. It first sends the message to one of the MDOs. Now two MDOs have the message, the proxy and the MDO, which just received the message from the proxy. Thus the second MDO can also participate in message propagation process. In the next step, both MDOs send the message to two other MDOs. Therefore the number of MDOs which have the message will be four. Then, all the four MDOs deliver the message to four other MDOs and this process continues until all MDOs receive the message. As can be seen, the number of MDOs which know the message is doubled after each step and each MDO receiving the message participates in message delivery. The mechanism does not assume any particular service in the underlying network and can be applied in all kinds of heterogeneous inter-networks.

2.3 Message Propagation Algorithm

In order to find the proper MDO to deliver the message in each step and to prevent double message delivery to an MDO, MDOs use an algorithm to determine the MDO which they should send the message to in each step of message propagation. The algorithm is shown in figure 2. The algorithm in each iteration works with a portion of the MDO list which is called *customized MDO list*. The customized MDO list is determined by two boundary indices and involves the MDOs which are between these indices in the original MDO list.

As mentioned, the proxy MDO receives the message first. The customized MDO list for the proxy in the first step of the message delivery is the original MDO list except itself. The proxy finds the median of its customized MDO list using the formula depicted in 4th step of the algorithm. Then, it sends the message and the boundary indices of the second half of the customized MDO list, which are calculated in the

6th step of the algorithm, to the median MDO, and makes it responsible for propagating the message among MDOs located between indicated indices in the list, using the same algorithm. In the next iteration, the proxy continues executing the algorithm using the first half of the customized MDO list as its new customized MDO list. The boundary indices of the new customized MDO list are calculated in the 5th step of the algorithm. Figure 3.a shows how a message is propagated among 16 hosts. The message delivery process in figure 3.a can be depicted as a special tree structure. Figure 3.b shows the message propagation tree in the sample system which presents a parent child relationship between MDOs.

Each MDO does the following steps after receiving a message.

Suppose the number of MDOs is n and the boundaries of the Customized MDO list are (a , b)

1. Get the message and the boundaries for the MDO list and calculate the customized MDO list.

2. If there is no boundaries

- a. If you are the proxy MDO and your position in the MDO list is p set the boundaries as $a = (p+1) \text{ MOD } n$ and $b = (p-1) \text{ MOD } n$.*
- b. Else finish*

3. If $b-a \text{ MOD } n < 2$ send the message to the MDOs which are at indices a and b and finish.

4. Find the median component of the customized list. Assume its position in the customized list is m then

$$m = (a + ((b-a) \text{ MOD } n)/2) \text{ MOD } n$$

5. Calculate the boundaries of the new customized list, which is the first half of the current list as follow:

$$\text{New\_}a = a \quad , \quad \text{New\_}b = (m-1) \text{ MOD } n$$

6. Send the message and the following boundaries to the MDO, which is at index m in the old MDO list.

$$a_{\text{for\_Median}} = (m+1) \text{ MOD } n \quad , \quad b_{\text{for\_Median}} = b$$

7. Set $a = \text{New\_}a$, $b = \text{New\_}b$ and go to step 1.

Fig. 2. MDO selection algorithm

When a MDO sends the message to all of its corresponding MDOs and finishes the iterations of message propagation among MDOs, it delivers the message to its co-located group member(s). MDOs which are leaves of the tree after delivering the message to their co-located group member(s) send acknowledge to their parent MDO. Each parent MDO after delivery of the message to its local group member(s) and receiving acknowledge from its entire child MDOs sends acknowledge to its parent MDO. The acknowledge indicates that all group members co-located with it and other MDOs in its sub-tree have received the message. For instance, an acknowledge for a message from Host12 indicates that all group members and MDOs on Host12, Host 13, Host 14 and Host15 have received the message.

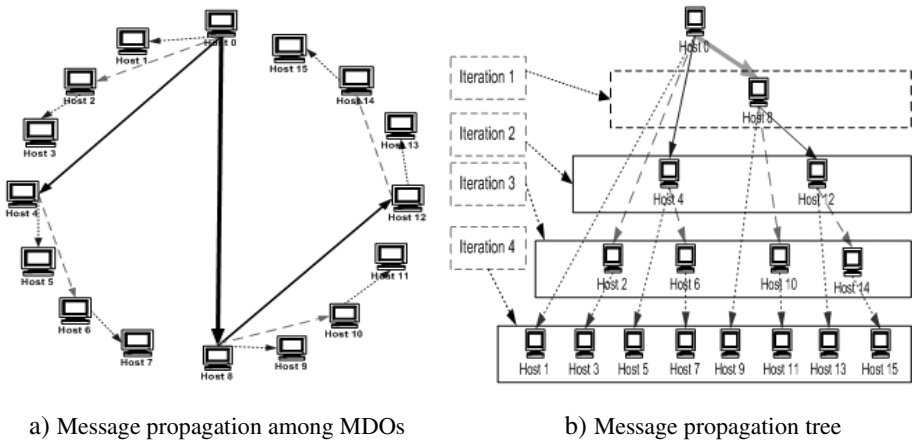


Fig. 3. Message propagation among MDOs

When the proxy MDO receives acknowledge from all of its child MDOs, it knows that all group members have received the message and there is no need to save the message in MDOs anymore. Then, the proxy MDO informs all other MDOs to delete the message from their message queue using a piggybacking method and the next incoming message. In other words, the proxy attaches the sequence number of the message to the next incoming message and propagates it among all MDOs. Each MDO after receiving the new message deletes the previously delivered message, which its sequence number is attached to the message, from its message queue. Then it continues the propagation of the new message using the mechanism.

3 Recovery from the Host Failure

The proposed group communication mechanism can detect the host failure in the system and recover itself from it. We suppose that the failure model of hosts in the system is *fail-stop* [2] which can be detected by other MDOs. Consequently, if a

MDO wants to send a message to another MDO on a failed host the sender can detect the failure. Host failure can be categorized as the following.

- 1. Failure before receiving a message.
- 2. Failure after receiving a message and before sending acknowledge for it.
- 3. Failure of the proxy MDO before receiving a message.
- 4. Failure of the proxy after receiving a message and before receiving acknowledges from its entire child MDOs.

For the sake of simplicity, we suppose that just one host fails in the system. In case of multiple host failure, the mechanism takes the same steps for recovery. The recovery mechanism uses acknowledges and timeouts to recover. After receiving a message and before sending it to its child MDOs, each MDO sets a timer. The timer value is calculated using the size of its first customized MDO list.

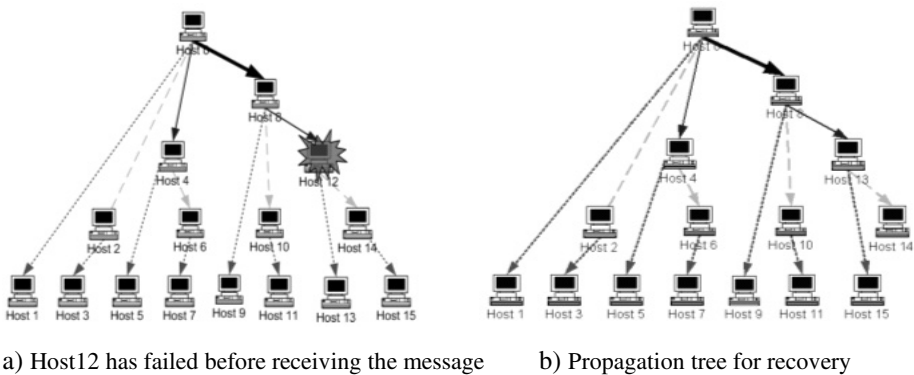


Fig. 4. Recovery from failure of host 12.

In the first category of the failure, when a host fails before reception of a message by its MDO, its parent MDO which wants to send the message to it detects its failure. The parent MDO first informs its parent about the failure and requests it to reset its timer. Each MDO after reception of the timer reset request resets its timer and sends the request to its parent MDO. This process continues until reaching to the proxy. Then, the parent of the failed MDO selects the neighbor MDO of the failed MDO, which is in the next index in the customized MDO list, as the new median and continues the process. It also informs the proxy MDO about the failure. The proxy MDO informs all other MDOs about the failure using the same message propagation mechanism. Each MDO after reception of the failure report first deletes the failed host from its MDO list and then uses the new list to participate in informing MDOs about the failure. Figure 4 shows the message propagation tree before and during the

recovery process from failure of host 12 in the previous example. The propagation tree after recovery is shown in figure 5.

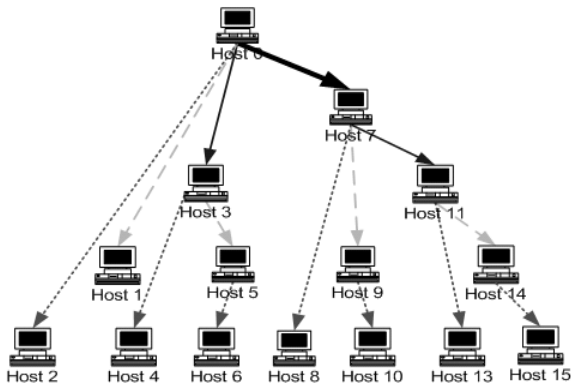


Fig. 5. Message propagation tree after recovery from failure.

For recovery from the second types of failures, the mechanism uses timers. As mentioned previously, before sending a message to its child MDOs, each MDO sets a timer. If the MDO does not receive acknowledge from its entire child MDOs before the expiration of the timer, it tries to detect which of the MDOs has not sent acknowledge. The MDO first sends a timer reset request to its parent as the previously described method. Then, it recovers from the failure using the same process described for the first type of failure.

Recovery from the proxy failure requires some different steps. If the host of proxy MDO fails before receiving a message the sender informs its local MDO about the failure and delivers the message to it. The MDO then chooses the second MDO in the MDO list, which is the neighbor of the proxy in the list, as the new proxy. Then, the new proxy deletes the failed MDO from its list and inform all other MDOs to delete the failed proxy from their MDO list using the same message propagation mechanism. Next, the message is sent to the new proxy by the MDO.

The last form of failure is detected by the child MDOs of the failed proxy. When the child MDOs want to send acknowledge to their parent MDO the failure of the proxy is detected. Then, the detector MDO chooses the second MDO in the list as the new proxy and informs all other MDOs about it using the previously described method. In the next step, the new proxy asks all MDOs to find out which messages have not been delivered to all group members and then resends them to all MDOs. All communications in these sections are done using the same group communication mechanism.

4 Some Characteristics of the Mechanism

Message Delivery Time. The proposed approach reduces multicast message delivery time in large scale systems. As mentioned before, the message delivery operation among MDOs is done in a parallel manner, which considerably improves multicasting speed. To calculate the maximum amount of time taken to disseminate a message among all MDOs, we can use formula (1).

$$\text{Maximum Time for Propagation Among MDOs} = \text{MMTT} * \log_2 (N_{\text{MDOs}}) \quad (1)$$

In the formula MMTT indicates the maximum message transfer time from one host to another and N_{MDOs} shows the number of MDOs in the system which is equal to the number of hosts that have group members. The maximum amount of time takes to deliver a message to all group members can be achieved by just adding message delivery time to group members by MDOs to the previously calculated amount.

$$\text{MaximumDeliveryTime} = \text{MMTT} * \log_2 (N_{\text{MDOs}}) + \text{LMDT} * N_{\text{Local\_Members}} \quad (2)$$

In formula 2, LMDT indicates the local message delivery time and $N_{\text{Local\_Members}}$ indicates the maximum number of group members in one host.

Open and Close Groups. The groups can be open or close. In an open group, objects, which are not group members, can send message to the group. In contrast, in a close group only group members can send messages to the group [2]. To support an open group, we make the proxy accessible from the outside. Then, messages can be sent to the proxy to be delivered to the group members.

Transparent Multicast. Transparent message multicasting is another property of our mechanism. In transparent multicast, the sender need not to know the group members and it just sends a message to the group and the mechanism will deliver the message to all group members [2]. This is a very important characteristic that makes implementation of the sender and the group members easy.

For the sake of simplicity, we supposed that each host sends one message at each time. By some changes in the mechanism the method can exploit multithreading capabilities of hosts to send more than one message simultaneously. For instance suppose each host can send 3 messages at the same time. Thus, each MDO instead of dividing its customized list into two halves by finding median; divides it into four sections and finds three MDOs to send the message. Consequently, message propagation among MDOs can be done in fewer steps.

We have tested our mechanism with 16 hosts and with one member on each host. The results have been compared to the mechanism that sequentially sends a message to all group members. On a 100Mbps LAN and with a 20KB message the proposed mechanism delivered the message to all group members in about 420 ms while the other one did the same task in about 1700 ms.

5 Conclusions and Future Work

We proposed a group communication mechanism for large scale distributed objects on heterogeneous inter-networks. The proposed mechanism disseminates message among group members in a considerably low time without any special assumption about the underlying network. Using special objects called Message Dispatcher Objects (MDOs), which are co-located with each group member; the mechanism parallelizes message propagation by distributing the propagation load among all hosts. Our proposed mechanism also detects host failure in the systems and recovers from it.

One of the main parts of the future work is devoted to issues such as membership protocols and recovery from other kinds of failures like network partitioning. More comprehensive evaluations and comparison of the mechanism with other similar mechanisms are planned to be done.

References

1. Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. : Group Communication Specifications: A Comprehensive Study. In *ACM Computing Surveys* 33(4), pages 1–43, December 2001.
2. G.Coulouris, J. Dollimore and T. Kindberg. : *Distributed Systems – Concepts and Design*, 3rd edition Addison-Wesley, 2001.
3. OMG. Fault Tolerant CORBA Specification. OMG document ptc/2000-04-04, 2000
4. Schiper, A and Rynal, M.: From Group Communication to Transactions in Distributed Systems. In *Communications of the ACM* 39(4), pages 84–87, April 1996.
5. Fekete, A. and Kedar, I.: A Framework for Highly Available Services Based on Group Communication. In *IEEE 21st International Conference on Distributed Computing Systems Workshops(ICDCS-21W 2001)*; the International Workshop on Applied Reliable Group Communication. April 2001
6. Goft, G and Yeager Lotem, E.: The AS/400 Cluster Engine: A Case Study. In *IGCC 1999 in conjunction with ICPP 1999*.
7. Birman, K., Friedman, R., Hyden, M. and Rhee, I.: Middleware Support for Distributed Multimedia and Collaborative Computing. In *Multimedia Computing and Networking (MMCN98)* 1998.
8. L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos.: Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4): 54–63, April 1996.
9. van Renesse,R., Birman, K. and Maffei,S.: Hourus: a Flexible Group Communication System. *Communications of the ACM*, 39(4), 76–83, April 1996.
10. Dolev,D. and Malki,D.: The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4), 64–70, April 1996.
11. Yair Amir and Jonathan Stanton.: The Spread Wide Area Group Communication System. Technical Report CNDS-98-4.Johns Hopkins University. 1998

Application of Catalogues to Integrate Heterogeneous Data Banks

Zsolt T. Kardkovács¹, Gábor M. Surányi², and Sándor Gajdos³

¹ Budapest University of Technology and Economics
Department of Telecommunications and Media Informatics
H-1117, Budapest, Magyar tudósok körútja 2., Hungary
`lodoktor@db.bme.hu`

² University of Karlsruhe
Institute for Program Structures and Data Organization
Am Fasanengarten 5, D-76128 Karlsruhe, Germany
`suranyi@ipd.uni-karlsruhe.de`

³ Hewlett-Packard Hungary,
H-1117, Budapest, Neumann János u. 1., Hungary
`sandor.gajdos@hp.com`

Abstract. The problem of integrating independent data banks is manifold. One of the arising issues is that data models of the same or related subjects vary among implementations. Therefore, identification of analogous, (fundamentally) identical or just similar concepts is a must. In this paper we deal with this problem. Based on preliminary descriptions of concepts, our novel integration method reorganises all data by means of a so-called catalogue. The strict mathematical basis of the catalogue enables 1) discovering the correspondence among concepts of the sources irrespectively of their language, 2) effective searching for exactly matching, parallel or alternative, most similar elements in the unified data bank. Our approach affords those databases the ability of transparently providing extended, personalisable services of better quality to the clients, which is highly demanded and applicable by modern web agents.

Keywords: heterogeneous databases, data integration, cataloguing, multilingual environment, relational data model.

1 Introduction

The proper integration of data banks, i.e. the integration of structured information sources is inevitable to provide high quality, up-to-date services on the Internet. The difficulty of integration lies in the dissimilarity of involved data sources. The manifestation of source dissimilarity may be:

1. architectural, that is data banks differing in organisation or specific products are used,
2. structural, meaning data models with different paradigms are employed (e.g. relational and object-oriented),

3. conceptual, that is concepts (attributes) have various semantics, what is reflected in their usage,
4. lingual, that is different words have the same meaning (e.g. synonyms, terms of multiple languages).

We consider the first an implementation problem. Therefore, we do not deal with it in this paper.

The second one can be easily surmounted if virtual integration is adequate: all data need to be temporarily converted into a common, expressive model. In this paper we utilise non-normalised relational schemata for this purpose.

Our research-work focuses on the 3rd and 4th problems. To deal with the third one, concepts, their relationships must be semantically interpreted. It is usually achieved by application of ontologies (see e.g. [7], [1], [2], [3], [5]). However, the development of ontologies is extremely difficult and time-consuming. Therefore, it is worth using semantically less expressive auxiliary information, e.g. so-called glossaries. This is a key feature of our method described in this paper.

Lingual dissimilarity is similar to the conceptual one. But it may require separate handling since ontologies usually cover the schema level only. As you will see, the glossary mentioned in the preceding paragraph fills both parts.

The ultimate purpose of integrated data sources is to provide a single system which is capable of answering queries based on all data originating from heterogeneous sources. In most cases these queries do not differ from the others submitted to the stand-alone systems. Recognition of similar elements across sources may be another task. If it is accomplished in pairs of entities, the time complexity is high. Our method supports this task natively, i.e. without additional processing after the data banks have been integrated. Furthermore, best-matching concepts are also determined although glossaries do not contain this information explicitly.

The paper is organised as follows. In the next section the notation is introduced and the formal description of all input data is given. Section 3 defines an initial data transformation. Section 4 deals with two relations which organise all the data. In Section 5 an efficient representation of these relations, the catalogue is introduced. The capabilities of catalogues, including their basic operations are discussed in Section 6. Section 7 presents an example for database integration using a catalogue. At the end conclusion is drawn.

2 Preliminaries

2.1 The Notation

Further on in this paper let us assume that the finite data stores to be integrated, denoted by $\mathcal{D}_1, \dots, \mathcal{D}_d$ ($d \in \mathbb{N}^+$), are represented over a common extended relational schema $\Omega = \{a_1, \dots, a_o\}$ ($o \in \mathbb{N}^+$). Ω is in ONF, i.e. each element of the relations over Ω contains a set of values for each attribute, $a_1, \dots, a_o$. The adjective *extended* refers to the fact that NULL values may occur as an attribute value in a relation over Ω . NULL values are useful in achieving a single, common

schema since values that are undefined in the original relations can be filled with NULL. Moreover, in a 0NF schema empty sets are legal attribute values. Thus, we will consider NULL values as empty sets.

$U(a)$ stands for the universe of the attribute a , i.e. the set of valid values of the attribute and

$$U(\Delta) = \bigcup_{a \in \Delta} U(a),$$

where Δ is a relational schema.

Possibly indexed capital letters from the beginning of the alphabet are used for elements of a relation (i.e. the tuples). Capital letters from the end of the alphabet (with or without an index) stand for variables over relations or attributes. The fields (i.e. attributes) of tuples are designated by possibly indexed lower case letters. The dot operator ($\cdot$) selects attribute values from the tuple written on its left side (projection). The names of the attributes are given plainly or as a set on the right side. The result is always a set of tuples of values from the corresponding universe, that is set values of the attributes are combined with each other every possible way. In graph-related formulae, the symbol v with or without index stands for a vertex of a graph or for an element of the vertex, not for an attribute.

In the following paragraphs many set relations are used and in the formulae logical implication is also needed. Thus, for clarity we note that $\supset$ always designates logical implication.

2.2 Total Glossaries

Besides the data stores, let us assume that there is a **total glossary** predefined. The role of this glossary is to describe concepts via enumerating their observable properties.

Example 1. The concept *sport car* could be described as follows:

- the engine volume of the element is *big* and
- the element has 4 wheels.

Of course, this definition is meaningful only if engine volume and number of wheels are contained in the schema.

Formally, let $\Theta = \{b_1, \dots, b_t\}$ ($t \in \mathbb{N}^+$) a 0NF relational schema, where the attributes (b_1 etc.) are the concept categories. Each concept category is derived from one or more attributes of Ω . This derivation is defined by a function

$$\chi: \Theta \rightarrow 2^\Omega,$$

where $\forall b \in \Theta \supset |\chi(b)| \geq 1$. Later we refer to

$$\chi': 2^\Theta \rightarrow 2^\Omega$$

also, which is defined as

$$\chi'(\Delta) = \{a | \exists b \ b \in \Delta \wedge a \in \chi(b)\},$$

where $\Delta \subseteq \Theta$.

$$\tau_b: \times_{a \in \chi(b)} U(a) \rightarrow U(b)$$

is another function parametrized by attributes of Θ . The *glossary* is the set of these τ_b ($b \in \Theta$) functions. A glossary is called *total glossary* if all its functions are total functions. This means that any combination of attribute values in Ω has its concept if a concept category is based on those attributes. In real-life scenarios it is easy to implement because each unused combination may represent itself as a concept also.

Example 2 (previous continued). Considering sport cars, the concept category is *car type*, *sport car* $\in$ *car type* and

$$\{\text{engine volume, number of wheels}\} \subseteq \chi(\text{car type}).$$

3 Primary and Secondary Tuples and Relations

The relations $\mathcal{D}_1, \dots, \mathcal{D}_d$ are also called **primary relations** and their elements are **primary tuples**. It is obvious that to every element (primary tuple) in any data store (primary relation) over Ω all their corresponding concepts can be assigned. These elements, the **secondary tuples** consisting of mapped concepts form another relation, the **secondary relation** over Θ .

Precisely, let

$$D_C = \langle D_{C,1}, \dots, D_{C,t} \rangle$$

be a *secondary tuple*, where C is an element of a data store, $\mathcal{D}$ is over Ω and

$$D_{C,i} = \{\tau_{b_i}(x) | \exists x \ x \in C. \chi(b_i)\} \quad (i = 1, \dots, t).$$

That is D_C contains the concepts C can be classified into via the total glossary, $\{\tau_b\}$ ($b \in \Theta$) based on the attributes of C . Then $\mathcal{C}_{\mathcal{D}} = \{D_C | C \in \mathcal{D}\}$ is the *secondary relation* associated with the primary relation $\mathcal{D}$.

Example 3. Imagine a primary relation containing data of cars. These data include “contains catalyst”, “engine volume” and “number of seats”. Then a secondary relation may contain the attributes “environment protective” and “car type”. The former is simply derived from the “contains catalyst” field, the latter from “engine volume” and “number of seats”. Its domain may be *sport car*, *van*, *truck*, *normal*. The secondary relation specifies which car from the primary relation belongs to which class in each of the two categories.

4 Substitutability

In business-to-business commerce catalogues specify products according to content standards, which, in turn, establish a hierarchy of product categories and specify the attributes of each category[4]. Catalogues used in our database integration method act similarly: they contain all data elements (as you will see in the next section), and the hierarchy is determined by the following two relations among the data elements.

Definition 1 (ε/m -substitutability). *Let C and D be elements of a relation over the schema Δ . Let*

$$\varepsilon: U(\Delta) \times U(\Delta) \rightarrow \{false, true\}$$

a transitive and reflexive function, a relation in the meaning used in set theory. C is substitutable by D concerning an attribute $m \in \Delta$, denoted by $C \preceq_{\varepsilon/m} D$ if

$$\varepsilon(C.m, D.m) = true.$$

Example 4. Let $\Delta = \{name\}$ and $U(\Delta)$ contain first names. For simplicity, we assume the relation R over Δ is in 1NF. Let ε be interpreted as the relation $\leq$ on strings that is an alphabetical ordering of string values. If for some $C, D \in R$ $C \preceq_{\leq/name} D$, then C stands before D in the alphabetical order. For example: $John \preceq_{\leq/name} Sam$.

The concept of substitutability can be generalised to a set of attributes as follows.

Definition 2 ($\varepsilon/\mathcal{M}$ -covering). *Let ε a function with the same properties described in the previous definition, $\mathcal{M} = \{m_1, m_2, \dots, m_i\} \subseteq \Delta$, a finite set of attribute names, let C and D be elements of a relation over Ω . C covers D concerning the function ε and the set $\mathcal{M}$, denoted $C \sqsubseteq_{\varepsilon/\mathcal{M}} D$, if*

$$\forall m \quad m \in \mathcal{M} \supset C \preceq_{\varepsilon/m} D.$$

Example 5. $\varepsilon/\mathcal{M}$ -covering has greater power of expression than ε -substitutability. For instance let $\mathcal{M} = \{name, info\}$,

$$C = \langle \{John, Sam\}, \{friends\} \rangle$$

and

$$D = \langle \{John, Sam\}, \{friends, brothers\} \rangle.$$

Then it is obvious that $C \sqsubseteq_{\subseteq/\{name, info\}} D$, and it means not only that C is before D in the order but, in this case, D is more informative about the relationship between the two persons than C .

Furthermore, if $\varepsilon = \subseteq$ and $\mathcal{M} = \Omega$, covering corresponds to the *type conformance* relation[6] in object-oriented modelling. If and only if C and D are entities and $C \subseteq_{\subseteq/\Omega} D$, all attributes of C occur in D . A 0NF model can not reflect types but a subset relation between the attribute values of the entities also holds. If there were types, that would fulfill the covariance condition, and contravariance is meaningless without input parameters.

Both primary and secondary relations and their schema are eligible for application in the ε/m -substitutability and $\varepsilon/\mathcal{M}$ -covering relations. Their ε functions can also be freely chosen. That means the relations among the tuples may vary significantly depending not only on the selected attributes but whether primary or secondary relations are considered.

In the rest of the paper ε is defined as $\subseteq$. As it will be seen, this ε function is quite practical since it naturally suits 0NF relations. Further, in this case the relations among primary and secondary tuples are similar to each other if their definition is based on corresponding attributes.

Theorem 1 (Correspondence of $\subseteq/\mathcal{M}$ -covering). *Let $C_1, C_2 \in \mathcal{D}$ where $\mathcal{D}$ denotes a data store and $\mathcal{M} \subseteq \Theta$. Then $C_1 \subseteq_{\subseteq/\chi'(\mathcal{M})} C_2$ implies $D_{C_1} \subseteq_{\subseteq/\mathcal{M}} D_{C_2}$.*

Proof. By definition $C_1 \subseteq_{\subseteq/\chi'(\mathcal{M})} C_2$ means

$$\forall a \ a \in \chi'(\mathcal{M}) \supset C_1.a \subseteq C_2.a. \quad (1)$$

Let $b \in \mathcal{M}$ a concept category and

$$x \in D_{C_1}.\mathcal{M}. \quad (2)$$

Because of (2) there exists a combination of attribute values in $C_1.\chi'(\mathcal{M})$ mapped to x by $\tau(b)$. Then (1) ensures this combination of attribute values also occurs in $C_2.\chi'(\mathcal{M})$. Since $\tau(b)$ is the same in both cases, $x \in D_{C_2}.\mathcal{M}$. That is, every element of $D_{C_1}.\mathcal{M}$ is element of $D_{C_2}.\mathcal{M}$, which is the definition of $\subseteq/\mathcal{M}$ -covering. $\square$

Similar proposition holds for $\subseteq/m$ -substitutability if $|\chi(m)| = 1$. The proof is implied by the previous theorem choosing $\mathcal{M} = \{m\}$.

5 Building Catalogues

The previously introduced relations among data elements are burdensome to handle. Catalogues provide a highly expressive way to represent these relations.

Definition 3 (Full $\varepsilon/\mathcal{M}$ -catalogue). *Let $G_{\varepsilon/\mathcal{M}} = \langle V, E \rangle$ be a graph, where V is a primary or secondary relation, $\langle v_i, v_j \rangle \in E$ if and only if $v_i \in V$, $v_j \in V$,*

$$v_i \subseteq_{\varepsilon/\mathcal{M}} v_j \text{ and } v_i \neq v_j.$$

$G_{\varepsilon/\mathcal{M}}$ is also called full $\varepsilon/\mathcal{M}$ -catalogue.

Apart from the missing loops (which would be present at all vertices), a full catalogue is equivalent to the $\varepsilon/\mathcal{M}$ -covering it is based on. In any full catalogue there may be elements that cover each other. Moreover the following proposition holds:

Proposition 1. *The strongly connected components of $G_{\varepsilon/\mathcal{M}}$ are cliques.*

Proof. Since ε is transitive by definition, if there is a directed path between two vertices there is a direct edge between them, too. That implies if two vertices reside in the same strongly connected component, i.e. there are two paths connecting them in both directions, there are also edges between them in both directions, which is the definition of being part of a clique. $\square$

From the user's point of view the elements of cliques are indistinguishable concerning the observed attributes by $\varepsilon/\mathcal{M}$ -covering. Therefore, it is reasonable to consider them as single entities.

Definition 4 ($\varepsilon/\mathcal{M}$ -catalogue). *Let us derive the simplified catalogue (or the catalogue for short), $G'_{\varepsilon/\mathcal{M}} = \langle V', E' \rangle$ from $G_{\varepsilon/\mathcal{M}}$ the following way: each clique of $G_{\varepsilon/\mathcal{M}}$ will be a vertex and two vertices, $v'_i \in V'$ and $v'_j \in V'$ corresponding to the cliques $C_i \subseteq V$ and $C_j \subseteq V$ are connected, i.e. $\langle v'_i, v'_j \rangle \in E'$ if and only if*

$$v'_i \neq v'_j \wedge \exists v_i \exists v_j \ v_i \in C_i \wedge v_j \in C_j \wedge \langle v_i, v_j \rangle \in E$$

and

$$\begin{aligned} \neg \exists v_i \exists v_j \exists v_k \ v_i \in C_i \wedge v_j \in C_j \wedge v_k \in C_k \wedge \\ \wedge \ C_i \neq C_j \wedge C_j \neq C_k \wedge \\ \wedge \langle v_i, v_k \rangle \in E \wedge \langle v_k, v_j \rangle \in E, \end{aligned} \quad (3)$$

where $C_k \subseteq V$ is a clique of $G_{\varepsilon/\mathcal{M}}$. The latter formula disallows short cuts in the catalogue.

Before proceeding we introduce three crucial property of catalogues.

Proposition 2. *$G'_{\varepsilon/\mathcal{M}}$ is a DAG.*

Proof (indirect). Because of the definition of the catalogue, if there is a circle in $G'_{\varepsilon/\mathcal{M}}$ any pair of tuples from both cliques corresponding to the connected vertices is also doubly connected in $G_{\varepsilon/\mathcal{M}}$. The previous proposition says they must be part of the same clique then. That means that they are included in the same vertex of $G'_{\varepsilon/\mathcal{M}}$, what contradicts the initial assumption. $\square$

Proposition 3. *The full catalogue can always be restored from its simplified catalogue, i.e. they contain the same information.*

Proof. The restore method is pretty straightforward:

1. The transitive closure is computed on the graph.
2. Loops are extracted into cliques of the tuples contained in the vertex.

To verify the correctness of the above method it is to be proven: during restore

- a. no edge is created which was non-existent in the original full catalogue,
- b. no edge is lost which was existent in the original full catalogue.

a: indirect. Let us assume a previously non-existent edge is created. This is possible only in step 1 because originally all edges exist in cliques of the full catalogue and only those edges are affected in step 2. Step 1 deals with transitivity, meaning although the elements corresponding the newly connected vertices are not connected in the full catalogue, an element from one of the vertices covers the other element from the other vertex because ε is transitive. Since the full catalogue is (almost) a direct representation of the covering relation, that should be reflected in it.

b: indirect. The restore method creates new edges, i.e. the missing edge is lost during reduction of the full catalogue to the simplified catalogue. Taking into consideration the definition of the simplified catalogue, only (3) may cause this. But it discards only "short cuts", they must be restored in step 1. $\square$

Proposition 4. *Any catalogue is minimal in the sense that no edge can be removed from it without altering its corresponding full catalogue.*

Proof. Removing an edge in $G'_{\varepsilon/\mathcal{M}}$ causes that edge to be missing in $G_{\varepsilon/\mathcal{M}}$ also. Otherwise that edge is restored at the first step. (The second step does not create edges between cliques.) In the first case the edge must be transitively implied, what contradicts (3) in the definition of $G'_{\varepsilon/\mathcal{M}}$. $\square$

The enumerated properties of catalogues show that simplified catalogues are efficient representations of full catalogues and their corresponding $\varepsilon/\mathcal{M}$ -coverings.

6 Operations on Catalogues

Catalogues are built to assist with retrieval of information. Therefore, the key operation on catalogues is search. Not crucial but for simplicity we assume that the catalogue consists of one component as an undirected graph, i.e. it has only one root. Let the search condition be a function, denoted by

$$k: \Delta \rightarrow 2^{U(\Delta)},$$

where Δ can stand for either Ω or Θ .

Algorithm (Simple Search in the Catalogue).

1. The current vertex is the root.
2. Select a function, m from k in which the current vertex differs from at least one of the next (i.e. connected) vertices.
3. Go down until the graph ends or the values of m in k matches the values of m of any element in the vertex.
4. Select, if exists, another differentiating function from k , m , and go to step 2. Otherwise, the algorithm ends returning all elements assigned to the current vertex.

Theorem 2. *The Simple Search algorithm always terminates within finite time.*

Proof. Because of Proposition 2 the algorithm can never return to a previously visited vertex. Moreover, the data stores are finite, that means only finite number of vertices exist. Within finite time all vertices can be visited. $\square$

N.B. the algorithm does not specify any particular order in which attributes are to be selected. It does not specify either which edge to follow if there are two or more proper directions in Step 2. It does not matter due to the following theorem.

Theorem 3. *If there are distinct vertices in $G'_{\varepsilon/\mathcal{M}}$ corresponding to v_i and v_j (primary or secondary) tuples, and $v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_j$, then there is a path from the vertex of v_i to the vertex of v_j .*

Proof. If there is an edge from the vertex of v_i to the vertex of v_j , the assertion is true. Then let us suppose that $v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_j$ but there is no such edge in $G'_{\varepsilon/\mathcal{M}}$. Since there is not any edge there, because of definition of $\varepsilon/\mathcal{M}$ -covering and $G'_{\varepsilon/\mathcal{M}}$, there must exist a vertex containing v_k so that $v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_k \sqsubseteq_{\varepsilon/\mathcal{M}} v_j$ and there is an edge from the vertex of v_i to vertex of v_k . Then the argument goes on recursively with v_k and v_j . As the graph is finite, the recursion will surely come to an end, thus there is a path. $\square$

In the previous proof any v_k vertex is proper if $v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_k \sqsubseteq_{\varepsilon/\mathcal{M}} v_j$ holds and there is an edge from v_i to v_k . As a result:

Theorem 4 (Container Theorem). *If there is a path between two vertices of $G'_{\varepsilon/\mathcal{M}}$ corresponding to v_i and v_j , all these paths contain all vertices of any v_k for which $v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_k \sqsubseteq_{\varepsilon/\mathcal{M}} v_j$ is true.*

Because every path to the requested element is equally proper, an advanced search may examine them in parallel. This also explains why more than one entry point (root) can exist in the catalogue. Parallel search can be quicker since the search stops when the element is found on one path.

The Container Theorem enables the definition of similarity search in the graph because the most similar elements to another reside next to it in the graph.

Definition 5 (Data Similarity). We use the notation $v \in G'_{\varepsilon/\mathcal{M}}$ to express the data element v is represented in $G'_{\varepsilon/\mathcal{M}}$. Let $v \in G'_{\varepsilon/\mathcal{M}}$ and let $\min(v)$ be

$$\{v_i | v_i \in G'_{\varepsilon/\mathcal{M}} \wedge v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v \wedge \forall v_j \quad v_j \in G'_{\varepsilon/\mathcal{M}} \wedge v_i \sqsubseteq_{\varepsilon/\mathcal{M}} v_j \supset v_j = v \vee v_j \not\sqsubseteq_{\varepsilon/\mathcal{M}} v\}$$

and let $\max(v)$ be

$$\{v_i | v_i \in G'_{\varepsilon/\mathcal{M}} \wedge v \sqsubseteq_{\varepsilon/\mathcal{M}} v_i \wedge \forall v_j \quad v_j \in G'_{\varepsilon/\mathcal{M}} \wedge v_j \sqsubseteq_{\varepsilon/\mathcal{M}} v_i \supset v = v_j \vee v \not\sqsubseteq_{\varepsilon/\mathcal{M}} v_j\}.$$

The most similar elements to v are

$$\max(v) \cup \min(v),$$

i.e. elements of $G'_{\varepsilon/\mathcal{M}}$ that cover v but are maximal with respect to it or are covered by v but are minimal with respect to that.

In case of primary relations, Data Similarity expresses which elements are the most akin to the one searched *irrespective* of their source. In a catalogue built from secondary tuples Data Similarity points out the closest concepts *without* analysing their definitions. If terms are enumerated in different languages in the glossary, the proposed method can successfully identify similar entities although they are using different languages. That means we reached our goal, the tasks identified in the introduction have been completed.

7 An Illustrative Example

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote two data stores of different employment agencies. They contain the important data of persons (name, experience, language and other skills). Concretely, $\mathcal{D}_1$ is:

name	experience	languages	skills/abilities
David	{Prolog}	{German}	NULL
Joe	{Oracle, XML}	{English, French}	{talented, intelligent, student}
Mary	{Java, C++, MSSQL}	{English, Spanish}	{gifted, associate}

And $\mathcal{D}_2$ is:

name	experience	languages	skills/abilities
Sam	{Oracle, Java}	{English, German}	{ambitious, ingenious, team player}
Sophie	{Prolog}	{Spanish}	{beginner}

The task is to select ideal persons for a job from both databases. The candidates should speak English, should have experience with *Oracle* and *Java*, and should be *young*, *ambitious* and *team player*.

As it can be seen, the domains of *experience* and *languages* are homogeneous. That is, the values are to be one-to-one mapped into a common secondary relation. However, values of *skills/abilities* need to be translated. For example this way:

ambitious = aggressive, ambitious, enterprising, go-ahead, pushing, up-and-doing, energetic, hardworking, indefatigable anxious, avid, eager, keen;

talented = apt, able, adept, astute, bright, capable, clever, expert, gifted, ingenious, intelligent, prompt, ready, savvy, sharp, skilled, skillful, smart, talented, teachable;

team player = collaborator, associate, colleague, confederate, fellow traveler, fraternizer, helper, partner, quisling, teammate, team player;

young = beginner, new, student, unpracticed, young.

Formally,

$$\chi(\text{norm. abilities}) = \{\text{skills/abilities}\}$$

and

$$\begin{aligned}\tau_{\text{norm. abilities}}(\text{aggressive}) &= \text{ambitious}, \\ \tau_{\text{norm. abilities}}(\text{enterprising}) &= \text{ambitious}, \\ &\vdots \\ \tau_{\text{norm. abilities}}(\text{student}) &= \text{young}.\end{aligned}$$

Then, the secondary relation is:

<i>name</i>	<i>experience</i>	<i>languages</i>	<i>norm. abilities</i>
David	{Prolog}	{German}	{}
Joe	{Oracle, XML}	{English, French}	{talented, intelligent, young}
Mary	{Java, C++, MSSQL}	{English, Spanish}	{talented, team player}
Sam	{Oracle, Java}	{English, German}	{ambitious, talented, team player}
Sophie	{Prolog}	{Spanish}	{young}

Based on these newly created records, a catalogue is to be built with $\varepsilon = \subseteq$ and $\mathcal{M} = \{\text{experience}, \text{languages}, \text{norm. abilities}\}$. The queries corresponding to the informal requirements are:

Q1 = *experience* minimum {Oracle, Java},

Q2 = *languages* minimum {English},

Q3 = *norm. abilities* similar {ambitious, talented, team-player, young}.

The catalogue returns the following results to the queries:

A1 = {Sam's data},

A2 = {Joe's, Mary's, Sam's data},

A3 = {Sam's, Sophie's data}.

Obviously, the requested answer is the common part of the sets returned.

Note that no single query with the same goal as **Q3** can be executed either in relational or in object-oriented databases just because the number of combinations of constants is proportional to the size of the database, i.e. to pose the query the content of the database should already be known.

8 Conclusion

We proposed a reorganising method and retrieval algorithm for integration of data originating from heterogeneous sources. The method is capable of discovering most similar data elements and concepts. Catalogues over primary tuples serve the former, while catalogues over secondary tuples serve the latter purpose. Concept similarity is language-aware also.

Because of its transitivity, the time complexity of catalogue building is essentially proportional to the complexity of an effective sorting algorithm and of a covering relation checking algorithm. We also showed that the several possible search queries are all parallelizable. One of them, similarity search not only offers an easily realizable, powerful extension to any existing database technology but it can ideally be applied by web agents as well.

References

1. Frans Coenen, Bridget Beattie, Trevor J. M. Bench-Capon, Michael J. R. Shave. Assessing Heterogeneity by Classifying Ontology Mismatches. *Formal Ontology in Information Systems*, (ed. N. Guarino). IOS Press, 1998, pp. 148–162.
2. Peter Fankhauser, Erich J. Neuhold. Knowledge Based Integration of Heterogenous Databases. *Proc. of IFIP Conference DS-5 on Semantics of Interoperable Database Systems*, 1992. Elsevier Science Publisher, North-Holland.
3. Eduardo Mena, Vipul Kashyap, Amit P. Sheth, Arantza Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. *Distributed and Parallel Databases*, 2000, 8(2), pp. 223–271.
4. Borys Omelayenko, Dieter Fensel. An Analysis of B2B Catalogue Integration Problems (Content and Document Integration). *Proc. of the International Conference on Enterprise Information Systems*, 2001.
5. M. Andrea Rodriguez, Max J. Egenhofer. Determining Semantic Similarity Among Entity Classes from Different Ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(2), pp. 442–456.
6. Unified Modeling Language Specification. Object Management Group. Version 1.5, March 2003.
7. H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner. Ontology-Based Information Integration — A Survey of Existing Approaches. *Proc. of the International Joint Conference on Artificial Intelligence Workshop: Ontologies and Information Sharing*, 2001.

Certificate Validation in Untrusted Domains

Omar Batarfi

School of Computing Science
University of Newcastle upon Tyne
Omar.Batarfi@ncl.ac.uk

Abstract. Authentication of other parties on the Internet using a Public Key Infrastructure (PKI) is still an open area for research. The current authentication method consists of building a certification path connecting a trust anchor¹ and the target, *a public key to be validated*. All the existing solutions cover the validation of a domain, *which could be hierarchical, single, cross-certification or bridge* [1], which holds the trust anchor and the target in the same domain of the CA root. Alternatively, the trust anchor and the target are in different domains but there is a link between the domains (cross-certification). This proposal presents a solution for domains that do not have a cross-certificate; therefore these domains could apply the proposed solution to authenticate their targets. In addition, a method for traversing and constructing a path will be presented.

1 Introduction

Enciphering information before it is transmitted via public networks is the only way to build secure communication. A Public Key Infrastructure (PKI) has an important role in achieving strong and trustworthy cryptography using a private and public keys (Public-Key Cryptography). In addition to that, public-key cryptography can be used to check another party's signature, and this function requires the establishment of secure contacts between two parties without prior contact.

A Certificate Authority (CA) issues the certificate for another CA or to an end-entity (*e.g., end-users, devices, Web servers, processes*) [2]. The following steps illustrate how the CA creates a certificate:

1. By using any public key algorithm such as keygen or RSA, the user creates his/her public-private key pair, then the user encrypts the private key and keeps it locally [3]. Finally, the user submits a certificate request to Registration Authority [RA] with his/her identity information, such as their name, address, telephone number and e-mail address.
2. In accordance with the section entitled IDENTITY VERIFICATION CHECK in the Certificate Practice Statement (CPS), the RA will validate the identity information in order to make sure it belongs to user, and the RA

¹ A trust anchor is a CA's public key that used by the party as trusted point to start path validation.

3. does this by checking mail, making a phone call, asking for identification, etc. According to result of this identity verification, the RA may approve or reject the request.
4. If the request is approved, the RA directs it to the CA to issue a certificate for the user.
5. The CA generates a certificate and signs it with its private key.
6. The RA sends a notification, *containing the serial number of the certificate*, to the user, and then the user retrieves his/her certificate from a repository by presenting the certificate's serial number to the RA Operator [4].

The certificate is a proof of identity on the Internet, so the user could claim that his public key is reliable, and a CA can vouch for the identity of the user. If there is another party (Bob) who wants to communicate with this user (Alice), Bob will ask Alice to show him her certificate. Bob will trust Alice's certificate if he trusts the issuer of the Alice's certificate; and Bob will start communicating with Alice. Alternatively, if Bob doesn't trust the issuer of Alice's certificate; then Bob needs to get the public key of the CA that issued Alice's certificate. Bob starts to extract information on Alice certificate, uses it to verify the other certificate, and Bob continues this process till he finds a trusted CA. This type of validation is called certification path processing.

Certification path processing is a complex process which could result in faulty output, and time consuming traversing. No standardization has been imposed to perform this process [2]. Certification path processing means establishing a path between the trust anchor and the target. During the construction of the path, all the certificates on the path must be checked and authenticated.

A *domain* is the environment that connects nodes (CAs and end-entities) together. When an authentication request is issued by a node (relying party)² to validate a target, the process of constructing certification path will start. The certification path processing is the mechanism for authentication of any certificate, but this processing is applicable for domains where:

1. The target and relying party reside on the same domain. See Figure 1.
2. The target and relying party reside on different domains, but these domains are connected together directly or indirectly. See Figure 2.

In the case where domains are not connected, there is no solution to verify the validity of the target's certificate. The aim of this proposal is to make the certificate validation possible even when the domains are disconnected. The proposed solution consists of building a certification path and validating the certificate as described in the following sections.

² Aas defined in [11] "A user or agent that relies on the data in a certificate in making decisions".

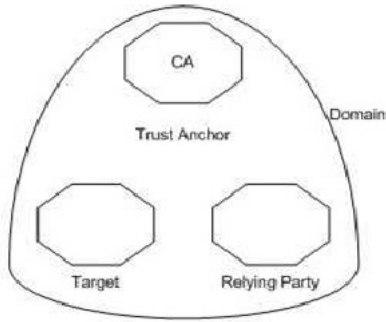


Fig. 1. Relying party and the target on the same domain

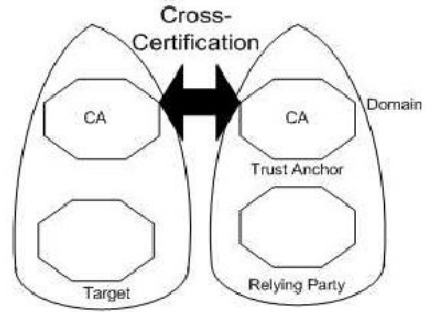


Fig. 2. Relying party and the target on different domains

2 Principles of Public-Key Infrastructure

PKI as stated in [5], “The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke Public-Key Certificates (PKCs)³ based on public-key cryptography”. This definition shows that the PKI is a cooperation of different entities and the main role of this cooperation is to establish a confidence relationship between its users.

2.1 PKI STANDARDS

It has been said that a Public Key Infrastructure is 80 per cent policies, procedures and processes, and only 20 per cent technology. Specifications of PKI policies have been clarified in written statements called standards. Currently, there are two open PKI standards: ITU-T Recommendation X.509 V3 and Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) [6], there are other standards but they either for internal use, such as the National Institute of Standards and Technology (NIST) [4], or where development of the standard has ceased, such as the Simple Public Key Infrastructure (SPKI) [7]. X.509 and PKIX are briefly discussed in what follows.

2.1.1 X.509

Recommendation X.509 defines a framework for a public-key infrastructure (PKI), including PKCs and Certificate Revocation List (CRLs) [5]. It defines the data objects that represent these certificates. Further, X.509 defines a framework for attribute certificates which includes definition of the information objects for Privilege Management Infrastructure (PMI), attribute certificates and Attribute Revocation List. X.509 also provides a framework for issuing, managing, using and revoking certificates.

³ As defined in [5] “A data structure containing the public key of an end-entity and some other information, which is digitally signed with the private key of the CA which issued it”.

Certificate extensions have been defined; and X.509 also defines a schema to store the PKI and PMI objects in a directory. A directory plays a great role in PKI and it makes uses of public key certificates and their attributes. Therefore, X.509 defines the method to facilities this using and to enable strong authentication. The X.509 standard describes two levels of authentication: simple authentication (*using username and password*) and strong authentication [*using an asymmetric (or public-key) cryptography*]. For providing secure services, only strong authentication should be used.

2.1.2 PKIX

The PKIX standard was proposed in [6] as an improvement for X.509, but when this goal was achieved, the PKIX work expanded beyond its goal and it has developed new standards for using X.509 on Internet. The first developed standard was the Certification Management Protocol (CMP) which includes a message protocol to initialize, certify, update, and revoke PKI entities; other standards including the Online Certificate Status Protocol (OCSP), Certificate Management Request Format (CRMF), Time-Stamp Protocol, Certificate Management Messages over CMS⁴ (CMC), Internet X.509 Public Key Infrastructure, Time Stamp Protocols and the use of FTP and HTTP for transport of PKI operations. Moreover, there are number of standards undertaken. PKIX has same functions as X.509 which facilitate the identification, authentication, access control and authorization functions on the Internet. PKIX also defines a profile for the certificate and CRL protocols. For improving the interoperability, the PKIX standard applied more restrictions to be followed between PKI clients [6]. Because PKIX does not require the use of an X.500 directory system [6], the PKIX use LDAP attributes to be integrated with certificates and CRLs for revocation use.

2.2 Certificate

A certificate binds the public key with the owner's identity, and its structure is roughly as the following [8]:

- Data structure (fields listed below)
- Signature Algorithm
- Signature

The basic certificate's fields include:

- Certificate Format Version: indicator of the X.509 version number.
- Certificate Serial Number: a unique numerical identifier for the certificate.
- Signature Algorithm Identifier: algorithm identifier of the digital signature algorithm.
- Issuer X.500 name: distinguished name (DN) of the Certification Service Provider.
- Validity Period: the dates and times for the start date and the expiry date.

⁴ Cryptographic Message Syntax.

- Subject X.500 Name: specifies the X.500 DN of the entity holding the private key corresponding to the public key identified in the certificate (see section 2.3).
- Subject Public Key Information: specifies the value of the public key owned by subject. And specifies the algorithm identifier specifying both the public key algorithm and the hashing function with which the public key is to be used.
- Issuer unique identifier: to link to the issuing CA name.
- Subject unique identifier: link to the subject name.

Version 3 of X.509 adds new fields to the basic certificate called extensions [8]. The main role of the extensions is to allow new fields to be added to certificate without modifying the certificate. Each extension consists of three fields: Type, value and Criticality.

- Type: contains an object identifier (*e.g. text string, date ...etc*).
- Value: contains the actual data for the extension.
- Criticality: a bit flag that indicates if an associated extension value is critical or non-critical. When an extension is flagged as critical, any application processing the certificate needs to process the value in the associated extension immediately [8]; if the application cannot process a critical extension, the application should reject the certificate.

2.3 Directory Service

The Internet is considered to be the largest distributed system; and therefore, to integrate the certificate with the Internet's other users, the certificate should be available to all system users on the Internet. For this reason, the idea of distributed repositories (directory) had been introduced. X.500 is a standard for the directory service by the International Union (ITU). The X.500 standard specifies both the structure of the directory information and protocols needed to access the directory information [9]. X.500 also defines a specification for a distributed directory based on hierarchically named information objects (directory entries) [9]. Each entry belongs to one or more object class (*e.g. country, person, organization*). Each entity contains a set of attributes which holds the object information, and at least one attribute's value specifies a name for an entry [8]. This name is known as distinguished name (DN) which is a unique identification. X.500 supports replication, access control and searching mechanisms.

The Lightweight Directory Access Protocol (LDAP) was designed to simplify the access to X.500 and to make the directory available to all clients (machines and applications) [8]. LDAP is subset of X.500, but LDAP goes beyond the original purpose and attempts to position itself as the only directory protocol required, and hence implies that the X.500 protocols are unnecessarily [9]. Using LDAP with X.500 is inefficient; the reason being the heavyweight nature of X.500. A site, *holding only the local data*, has to bring up a full X.500 service before the user could use LDAP to access these data. A new protocol has been developed called *slapd*, a stand-alone LDAP server which has its own local database [8].

2.4 Certification Path

In [10], the Certification path is defined as “An ordered sequence of certificates that, together with the public key of the initial object in the path, can be processed to obtain that of the final object in the path”. There is no standardized method for retrieving the certificates constituting a certification path in the PKI standard [11]. To help the process of constructing the certificate path, a number of attributes and extensions have been defined, for example, Authority Key Identifier (AKID) and Subject KEY Identifier (SKID). Moreover, the certificate attributes such as cACertificate, crossCertificatePair and userCertificate help in constructing a certificate path. X.509 defines a PkiPath attribute that should store the certification path (partial or complete), but no applications that make use of the pkiPath attribute [2]. An extension called Authority Information Access (AIA) has been defined to indicate how to access the CA’s information and services, and the Subject Information Access (SIA) has been defined to indicate how to access the subject’s information and services [2]. Two directions of the certification path construction are possible - either forward or reverse path direction. Forward direction means building the path from the target certificate to a recognized trust anchor, and conversely, reverse direction means that the process will start with a recognized trust anchor certificate and move towards the target certificate. Forward direction is more efficient to be applied in an hierarchical domain and reverse direction is best suited for a distributed domain [12]. A number of factors make the certification path construction inefficient or time consuming:

- Multiple paths between the trust anchor and the target certificate; this could happen when more than one candidate path between the trust anchor and the target certificate meets all of the certification evaluation criteria equally and no specific criteria exist for one candidate path to be preferred over another [2].
- Loops that lead to the already verified certificates; this could be happen when DN appears more than once, and the reason for this could be [12]:
 - A CA with a same DN and two different public keys.
 - Two different CAs with same DN.
- Incomplete information in a certificate’s attributes and extensions.

3 Validation Processing Methods

Validation processing methods are composed of two steps [6]:

- Determination: which means accessing a repository retrieve the certificate and the construction of the path.
- Validation: which means making sure that each certificate in the path has its integrity, is within its validity period; and has not been revoked.

In what follows, five existing proposals for validating the certificate within a trusted domain are presented, then, a new method for validating the certificate in an untrusted domain is proposed.

3.1 Certificate Chains

A certificate chain is described in [6] as a structure that contains all the certificates, *from the CA immediately below the root CA to the user*, signed and wrapped by the root CA of the hierarchy. The client stores all self-signed root CA certificates that needed in order that he/she can authenticate target of different hierarchies. Path processing with certificate chains is easy to perform, because the determination process is performed at the time of certificate-chain signing and wrapping, therefore, the validation mechanism is only needs to be applied.

3.2 Hierarchy Graphs

This method, discussed in [6], is a program that resides on the client computer which checks and stores all local CA hierarchy to which its user belongs before the path construction begins. The hierarchy is directed graph where the nodes represent CAs and the arcs are the certificates that interconnect those CAs. Nodes are inserted into the graph after its integrity has been checked, starting with the CA who issued the target certificate and the insertion process continues until an intersection node with the local hierarchy graph is found. After the determination task ends, the path validation is performed. The proposed algorithm is designed to find the shortest path among the multiple alternative paths.

3.3 Certification Path Validation Service

In [6], it is proposed that a data certification server (DCS) is responsible for the path processing and the DCS does this by validating the signature and the provision of updated certificate status information. For authentication, the client sends an authentication request to the DCS, together with the target certificate. The DCS will do validation for the target's certificate by retrieving all necessary certificates from distributed repositories. The DCS will determine and validate the path and send a token to the client. This token will allow several transactions for a limited time, without requiring the authentication process to be repeated. As we see the client implementation is simple because all the processing is performed in the DCS.

3.4 Modified LDAP Server

A repository access server, *as described in [6]*, will determine the path after it receives a request from a client; then it sends the path to the client including all the certificates that comprise it. The client is responsible for validating the received path. The LDAP protocol is the tool for accessing the repository.

3.5 Dynamic Path Determination

The dynamic path determination, as stated in [6], allows a root CA only to issue cross certificates to the root CAs of other hierarchies and no trust is to be transferred between such root CAs; and therefore, only one certification path is possible. The path is constructed by retrieving each certificate from a repository via LDAP; and a local list is called a local trust chain is stored in the client environment. The local list will assist in the process of certification path verification. The local trust chain contains all certificates that constitute the path between the user's issuing CA and its hierarchy root CA; including the latter. After the path determination process is complete, PKIX path validation algorithm is applied.

4 The Proposed New Method

In real life the physical identity is the only way to authenticate others, but the degree of the validation surely will affect success in communication between the parties. Typically, this philosophy does not apply on the Internet. For this reason, the principle of a third party, CA, has been introduced to perform the identity validation. Therefore, CA issues signed certificates to assure the validity of party's identity.

This proposal proposes a solution for validating the target identity if he/she is in a domain that differs from the relying party's domain and these domains are disconnected.

The proposed solution will help the root CA, *which is in the same domain as the relying party*, to make a decision to accept or reject the target's certificate. Constructing the certification path is required for validating the certificate with attributes such as expiration date, revocation status and a credibility path to its superior CAs. Also, the proposal suggests a method for constructing the certification path in an efficient manner. Before we explain the proposed solution, an essential step is to outline some policies which should be considered first:

Policy

1. Domains are not connected together.
2. Root CA is the only party that has the right to give the decision about the validation of the target's certificate.
3. No trust will be placed on any CA in the target domain.
4. There is no transitivity of trust between members of another domain.
5. The target's certificate must be validated in its domain first.

The proposed solution consists of two parts, and these parts are mandatory to complete the solution. The first part will carry out the determination and validation process in the target domain, and the second part will encapsulate the target's certificate to validate it in the relying party's domain.

4.1 Ants Traversing and Validation (ATV) Method

The proposed solution for constructing the path tries to simulate the ants method for finding food. Ants send a scout to find food, and the scout returns to the colony when he finds food and transmits the information to foragers [13]. In this proposal, the ants model will be altered in order to continue the role of the scout with foragers. The extended method to be applied as a basis for ATV solution is as follows:

1. The Scout starts the process of constructing the certification path by tackling the target's certificate and extracting its information from the repository through LDAP.
2. The information will be validated to check the status of the target's certificate:

If the certificate is invalid for any reason such expired; the ATV will end here and a message will be sent to relying party said "no accreditation can be placed on the target".

3. The Distinguished Name (DN) attribute will be extracted to determine the certificate's issuing CA.
4. Scout will move to the CA's certificate.
5. Foragers will move to the previous scout position.
6. Foragers will look to find any links connect to their position.
 - a. If there links, the foragers will move to the other positions.
 - b. If there are no links, they will stay in their position.
7. The scout starts to communicate with foragers to detect the suitable path.

Suitable path will be defined on some attributes that are defined before the process of construction takes place; attributes such as user-constrained-policy-set.

8. If the current position of the scout is suitable one (i.e. that there is an intersection in the attributes between the scout and one of the foragers):
 - a. The scout's certificate will be check to see if it self-signed or not
 - b. If self-signed, then a message will be sent to the relying party said that "the target certificate is correct".
 - c. The certificate will be added to the path.
 - d. The ATV will stop.
9. If the scout position is not the suitable one, then the scout will move to a suitable one which is held by one of the foragers.
10. The path will be modified to include the new certificate and the process will continue again from the step 3.

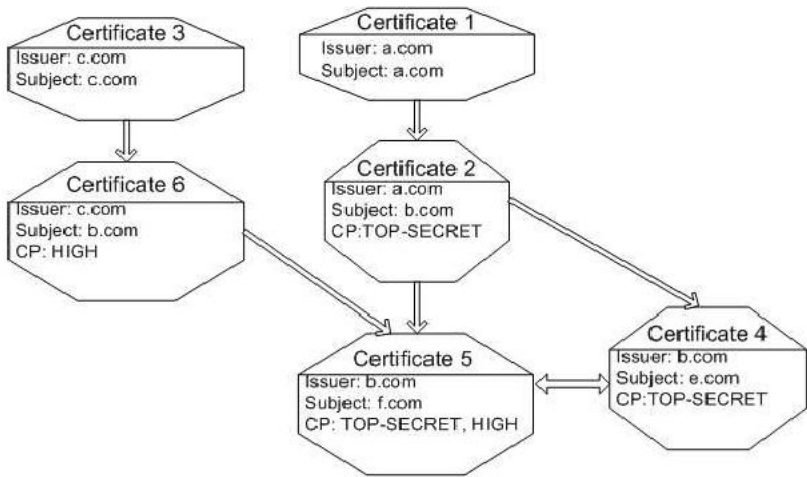


Fig. 3. ATV Processing Scenario

Example

To illustrate the concept of ATV, Let us assume that a user (*relying party*) asks to validate target's certificate which is certificate 5. The relying party could specify a set of initial policies that he/she finds acceptable; and after a valid path has been built, all the certificates on the valid path must contain these policies. The relying party defines these policies in the user-constrained-policy-set attribute [12]. In this example the user-constrained-policy-set is set to HIGH. The scout starts the process of extracting and validating the data of certificate 5. After the scout assures that certificate 5's data is valid, DN attribute will be extracted and the scout will move to certificate 2. Let us assume that certificate 5 has different public keys but with the same DN. The foragers will move to certificate 5, and the foragers discover that certificate 5 has three links; because the link toward certificate 2 is being held by scout, the forages will move to certificates 4 and 6. The scout will start to communicate with the foragers to determine from the certificate that its CP intersects with the user-constrained-policy-set. The scout finds that the certificate 2 is not the appropriate certificate because TOP-SECRET has no intersection with the user-constrained-policy-set. Therefore, the scout will move to certificate 6, and finds that certificate's CP contains HIGH. Because certificate 6 is not self-signed, the scout will move to the certificate 3 and the partial valid path C5>C6 will be considered. Certificate 3 is a self-signed certificate, so the validate path is C5>C6>C3 and the certificate 5 will be considered authenticated; consequently, the ATV will stop.

4.2 Encapsulate Validation (EV)

EV is the solution for authentication of the target's certificate that resides in a different domain from the domain that holds the trust anchor. Moreover, these domains are disconnected. The EV solution will get around this problem by transferring the target's certificate to the relying party's domain. Also the target's certificate will be en-

capsulated by the policy which has been applied in the relying party. EV solution steps are performed in the following manner:

1. After making sure that the target's certificate is valid in its domain, the certificate will be transmitted to the relying party domain.
2. The target's certificate will be connected directly to the Root CA
3. The policies that are acceptable and recognized (e.g. CPS and CP) in the domain will be applied on the target's certificate.
4. If there is a need for using the target domain's policy then this will be used.
5. After completing the steps 3 and 4, the root CA will be able to issue the decision about the credibility of the target, and a reply will be sent to the relying party.

4.3 Key Issues

4.3.1 Security

There are numbers of security concerns should be taking in consideration; those concerns include spoofing, impersonating and altering information in order to deceive, such as bogus policies. Those issues are real threats for any solution and could lead to harmful affect.

4.3.2 Adjacent Domains

All domains that have previous experience with target's domain could offer valuable information in grading the target certificate. The offered information should be careful investigated for making sure that it is valid evidence.

4.3.3 Target Certificate Paths

Building more paths from target certificate will increase in the strength of trust and confidence in target certificate; this concept is built on the saying that the chain is strongest as weakest. The trust here will be a cumulate of the trust in those paths of the target certificate.

4.3.4 Liability

High liability for the proposed solution should be grantee, and this target will be achieved by dealing properly with all expected shortcomings.

5 Conclusion

This proposal has introduced a solution to facilitate the validation in the case where there is an untrusted domain. All the solutions that currently exist are considered to be trusted domains because there is some sort of collaboration between the relying party's domain and the target domain. The proposed solution is based on two successive parts (ATV) and (EV); and when those parts totally applied, the relying party will have knowledge about the target's validity.

A new method for traversing and constructing the certificate path (ATV) is expressed in this proposal, and an example of ATV is used to illustrate the application of the method. Some issues, *related to the proposed solution*, which need further investigation have been listed.

References

1. Komar, B., *Troubleshooting Certificate Status and Revocation*. 2001: p. 64.
2. Lloyd, S., *understanding Certification Path Construction*. 2002: p. 14.
3. Olson, D., *Certificate Authority issues*. 2001: p. 7.
4. Xenitellis, S., *The open-source PKI Book: A guide to PKIs and open-source Implementation*. 2000.
5. UNION, I.T., *ITU-T RECOMMENDATION X.509 | ISO/IEC 9594-8: "INFORMATION TECHNOLOGY - OPEN SYSTEMS INTERCONNECTION - THE DIRECTORY PUBLIC-KEY AND ATTRIBUTE CERTIFICATE FRAMEWORKS"*. 2001: p. 162.
6. Alexandre R. SILVA, M.A.S., *Pequi: A PKIX Implementation for Secure Communication*. p. 11.
7. *SPKI (Simple Public Key Infrastructure)*. 2000: p. 33.
8. Commission, T.E., *Guidelines, Methodologies and Standards to set up a CA for Digital Signatures*. p. 103.
9. Ltd, D.C., *DIRECTORY SERVICES - THE ROLE OF LDAP AND X.500*. p. 6.
10. S. Chokhani, W.F., R. Sabett, C. Merrill, S. Wu, *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*. 2003: p. 76.
11. A. Arsenault, S.T., *Internet X.509 Public Key Infrastructure: Roadmap*. 2002: p. 57.
12. Yassir Elley, A.A., Steve Hanna, Sean Mullan, Radia Perlman, Seth Proctor, *Building Certification Paths: Forward vs. Reverse*. p. 8.
13. Zhanna Reznikova, B.R., *USING INFORMATION THEORY APPROACH TO STUDY THE COMMUNICATION SYSTEM AND NUMERICAL COMPETENCE IN ANTS*. p. 8.

Author Index

- Alonso, Alejandro 385
- Bacon, David F. 466
- Barchino, Roberto 166
- Bartoli, Alberto 3, 900
- Batarfi, Omar 1057
- Beebee, William Jr. 550
- Benowitz, Edward G. 490, 497, 508
- Bernardeschi, Cinzia 425
- Bernat, Guillem 411
- Bilke, Alexander 5
- Bizer, Christian 816
- Bo, Jan De 253
- Bollella, Greg 534
- Borg, Andrew 397
- Brodie, Michael 34
- Bübl, Felix 5
- Burns, A. 360
- Cai, Hao 319
- Calabrese, Cosimo 900
- Candan, K. Selçuk 871
- Capitani di Vimercati, S. De 741
- Carissimi, Alexandre S. 520
- Carr, Harold 109
- Carro, Luigi 520
- Catarci, Tiziana 32
- Cattaneo, G. 960
- Cattaneo, Giuseppe 975
- Catuogno, Luigi 975
- Ceravolo, Paolo 732
- Cha, Guang-Ho 26
- Chang, Elizabeth 151, 223, 235, 845
- Cheng, Perry 466
- Chockler, Gregory 885
- Chong, Cheun Ngen 779
- Cilardo, A. 960
- Combi, Carlo 22
- Corbett, Dan 24
- Corsaro, Angelo 318, 564
- Coupaye, Thierry 1
- Courtiait, Jean Pierre 16
- Cuppens, Frédéric 754
- Cytron, Ron K. 318
- Damiani, Ernesto 20, 703, 845
- Delgado, Jaime 621
- Deray, Tom 137
- Deridder, Dirk 35
- Devanbu, Premkumar T. 926
- Díaz, Manuel 7
- Dillon, Tharam S. 151, 223, 845
- Draheim, Dirk 267
- Dulong de Rosnay, Melanie 635
- Eckstein, Rainer 816
- Elgedawy, Islam 1004
- Etalle, Sandro 779
- Fargetta, Marco 437
- Farkas, Csilla 718
- Feizabadi, Shahrooz 550
- Feng, Ling 704
- Fernández, Fernando 175
- Fernández-Medina, E. 741
- Francesco, Nicoletta De 425
- Franconi, Enrico 32
- Fresco, Roberto 98
- Gajdos, Sándor 1045
- Gal, Avigdor 18
- Gallego, Isabel 621
- Gançarski, Stéphane 14
- Gangemi, Aldo 607
- García, Elena 152
- García, Roberto 621
- Gardner, William 223
- Garrido, Daniel 7
- Golani, Mati 18
- Gomes, Roberta Lima 16
- Gong, Xiaofeng 807
- Graba, Lee 899
- Güntzel, Kai 70
- Gupta, Deepak 793
- Gutiérrez, José María 166
- Gutiérrez de Mesa, José Antonio 166
- Hartel, Pieter H. 779
- Heese, Ralf 816
- Higuera-Toledano, M. Teresa 385, 479
- Hofreiter, Birgit 829
- Hogben, Giles 579, 859

- Hoyos-Rivera, Guillermo J. 16
 Hu, Erik Yu-Shing 411
 Huang, Wanjun 950
 Huang, Wen-Shi 197
 Huemer, Christian 829

 Iglesias, Ana 175

 Jafarpour, Hojjat 1036
 Jalote, Pankaj 793
 Janecek, Paul 185
 Jank, Klaus 57
 Jarrar, Mustafa 578, 594
 Jayaratna, Nimal 235
 Johannesson, Paul 690
 Jonker, Willem 704

 Kabilan, Vandana 690
 Kalogeras, Athanasios P. 335
 Kapnoullas, Terry 845
 Kardkovács, Zsolt T. 1045
 Karran, Tereska 123
 Kerremans, Koen 662
 Kingston, John 648
 Klein, Andrew 451
 Kodali, Naren 718
 Kriegel, E. Ulrich 5
 Kumar, Atul 793

 Leary, Richard 675
 Leenheer, Pieter De 675
 Leicher, Andreas 5
 Li, Peng 550
 Llorente, Silvia 621
 Loh, Krystal 534
 Lorentz, Kai 335

 Malhotra, Manish 83
 Malkhi, Dahlia 885
 Martínez, Paloma 175
 Martini, Luca 425
 Masarin, Giovanni 3
 Mascio, Tania Di 32
 Maverick, Vance 3
 Mayer, Christopher B. 871
 Mayo, José P. 385
 Mazzeo, A. 960
 McKendry, Graham 534
 McMullen, Andrew 899
 Meersman, Robert 34, 253, 594

 Meinel, Christoph 950
 Merimovich, Barak 885
 Michailovsky, Eugeny 18
 Miège, Alexandre 754
 Miguel, Miguel A. de 385
 Montresor, Alberto 900
 Muro, Etienne Antoniutti Di 912

 Newman, Julian 807
 Nguyen, Thao Thi 235
 Niessner, Albert F. 490, 497, 508
 Nilsen, Kelvin 371, 451

 Oberhauser, Roy 57
 Oberle, Daniel 28
 Oliboni, Barbara 20, 22

 Pape, Cécile Le 14
 Pascucci, Giuliano 306
 Pederiva, Andrea 98
 Perry, Stephen 11
 Petrillo, Umberto Ferraro 975
 Pitsilis, Georgios 1031
 Poggi, Agostino 989
 Prayati, Aggeliki S. 335
 Prica, Milan 900
 Prisco, Alessandra 607
 Prochazka, Marek 1
 Pu, Pearl 185
 Puder, Arno 292
 Pun, Thierry 769

 Quintarelli, Elisa 20, 22

 Rabinowitz, David 885
 Rajan, V.T. 466
 Ravindran, Binoy 550
 Reis, André I. 520
 Ribeiro-Justo, George R. 123
 Richards, Debbie 28
 Rinard, Martin 550
 Romano, L. 960
 Rosa, Leomar S. Jr. 520
 Roth, Uwe 950
 Rouvoy, Romain 1
 Rugaimukamu, Dickson M. 690
 Ruiz, G. 741
 Ryan, Caspar 11
 Ryan, Henry 675
 Rytsar, Yuriy 769

- Sabou, Marta 28
- Saggese, G.P. 960
- Sagri, Maria-Teresa 607
- Salaun, Anne 578
- Sangam, Venkatesh 871
- Santoro, Corrado 318, 564
- Santucci, Giuseppe 32
- Schoeberl, Martin 346
- Schwab, Christian 335
- Shin, Hyoseop 46
- Sicilia, Miguel-Ángel 152, 210
- Soetens, Peter 35
- Spadaro, Sandro 306
- Spyns, Peter 253, 675
- Stefano, Antonella Di 437
- Steve, Geri 607
- Surányi, Gábor M. 1045
- Tanca, Letizia 20
- Tangermann, Marcus 335
- Tari, Zahir 83
- Temmerman, Rita 662
- Terziyan, Vagan 279
- Tessaris, Sergio 32
- Thomas, Vicraj 899
- Tiscornia, Daniela 607
- Tolksdorf, Robert 816
- Tolosa, Ruth 385
- Tomaiuolo, Michele 989
- Tramontana, Emiliano 437
- Tsai, Tse-Ming 197
- Tummers, Jose 662
- Unland, Rainer 1017
- Vakalis, Ioannis 579
- Valduriez, Patrick 14
- Vandenberghe, Wim 648
- Verheyden, Pieter 137
- Verlinden, Ruben 594
- Visconti, Ivan 975
- Vitaglione, Giosuè 989
- Voloshynovskiy, Sviatoslav 769
- Volz, Raphael 28
- Wagner, Flávio R. 520
- Weber, Gerald 267
- Wellings, Andy J. 319, 360, 397, 411
- Wijesekera, Duminda 718
- Wilikens, Marc 579
- Wohlstadter, Eric A. 926
- Wong, Siu Keung 235
- Wozenilek, Thomas 534
- Yang, Yong-Fang 197
- Yazdani, Nasser 1036
- Yoneki, Eiko 934
- Yu, Han-Kuan 197
- Zheng, Jijuan 30