Lecture Notes in Computer Science 2641
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Peter J. Nürnberg (Ed.)

Metainformatics

International Symposium, MIS 2002
Esbjerg, Denmark, August 7-10, 2002
Revised Papers

Springer

Volume Editor

Peter J. Nürnberg
Aalborg University Esbjerg
Department of Computer Science
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
E-mail: pnuern@cs.aue.auc.dk

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.4, H.5.1, H.5.4, H.3, D.2, c.2, I.2, K.4, I.7, F.3, E.2, J.5

ISSN 0302-9743
ISBN 3-540-40218-7 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10932612 06/3142 5 4 3 2 1 0

Preface

This volume contains the final proceedings for the Metainformatics Symposium 2002 (MIS 2002). This event was held during 7–10 August on the campus of Aalborg University Esbjerg, located in Esbjerg, Denmark. MIS is at once both an extension of the previous Structural Computing (SC) Workshop series (see LNCS vols. 1903 and 2266 – available from Springer-Verlag – for the proceedings of SC2 and SC3), and a broadening of the themes of that workshop series in several new and exciting directions. I hope you find the work presented in these proceedings as interesting and exciting as I have.

I would like to thank Dr. Siegfried Reich and Mrs. Britta Jensen for their help in compiling these proceedings. Also, this volume would not have been possible without the support of Springer-Verlag, Heidelberg. In particular, I would like to thank the executive editor of the LNCS series, Mr. Alfred Hofmann.

October 2002

Peter J. Nürnberg
MIS 2002 Program Chair

Table of Contents

Introduction to MIS02 1
Peter J. Nürnberg

Building Metainformatical Bridges 6
Peter J. Nürnberg

Lessons Learned with the Construct Development Environment 9
Uffe Kock Wiil

A Meta-modeling Environment for Cooperative Knowledge Management . . 18
Jessica Rubart, Weigang Wang, Jörg M. Haake

Personalization in Knowledge Management 29
Klaus Tochtermann

Babylon Bookmarks:
A Taxonomic Approach to the Management of WWW Bookmarks 42
Nikos Karousos, Ioannis Panaretou, Ippokratis Pandis, Manolis Tzagarakis

Approaching Wordnets through a Structural Point of View 49
Dimitris Avramidis, Maria Kyriakopoulou, Manolis Tzagarakis, Sofia Stamou, Dimitris Christodoulakis

Software Development for Dynamic Systems 58
Darren Dalcher

Containment Modeling of Content Management Systems 76
Dorrit H. Gordon, E. James Whitehead Jr.

Smart Active Object: A New Object-Oriented Programming Paradigm
for Developing Multithreaded Applications 90
Yan Chen, Xinyuan Fan, Jian Jiao, Dongmin Wang

Implications of the View of Programs as Formal Systems 100
Bertil Ekdahl

In Search of a User Base: Where Are the B's? 112
David L. Hicks

Reconciling Versioning and Context in Hypermedia Structure Servers 118
Jon Griffiths, David E. Millard, Hugh Davis, Danius T. Michaelides, Mark J. Weal

Metadata Usage in Multimedia Federations 132
Mark Roantree, Damir Bećarević

Data Mining Using Links in Open Hypermedia 148
Dragos Arotaritei, Peter J. Nürnberg

EMMOs – Enhanced Multimedia Meta Objects 155
Sunil Goyal, Siegfried Reich, Wernher Behrendt

A UML Profile for Agent-Based Development 161
Amir Zeid

Widening the Configuration Management Perspective 171
Lars Bendix

Web Services in ERP Solutions: A Managerial Perspective 177
Bostjan Kezmah, Ivan Rozman

An Infrastructure to Manage Resources 180
Frank Wagner

Author Index 187

Introduction to MIS02

Peter J. Nürnberg

Department of Computer Science
Aalborg University Esbjerg
Niels Bohrs Vej 8
DK-6700 Esbjerg, Denmark
pnuern@cs.aue.auc.dk
http://cs.aue.auc.dk/ pnuern/

1 Background

The International Symposium on Metainformatics (MIS '02) was held from 7.-10. August 2002, in Esbjerg, Denmark. There were 27 registered attendees from 11 different countries. A total of 15 papers (11 full papers, 2 short papers, 2 position statements) were presented. Authors of papers were allowed to revise their submissions based on the feedback they received at the sympsoium and submit them for a second review cycle. This volume is the result of that rigorous two-stage review process.

Metainformatics concerns work about computer science – about developers, computer users, or those influenced by computer technology. Metainformatics is about finding the common ground shared by researchers and practitioners in many different computer science areas, who may use similar methods to achieve different ends.

As pointed out by Douglas Engelbart [1], there are different levels of computer science work. Consider the software engineering and programming languages areas. Front-line (or A-level) programmers may use "off the shelf" applications, such as compilers or IDE's, to do their work. These applications are in turn constructed by B-level developers – literally, developers of tools for other developers. The methodologies and frameworks used by these B-level developers are developed by C-level developers.

The metainformatics symposium brought together researchers from a variety of fields including computer science, social sciences and the humanities that practice or are interested in B- and C-level work to share and discuss ideas, to find common themes in these "meta"-level pursuits, and to learn from one another's experiences.

The call for participation for MIS '02 included a wide list of topics, with special consideration given to submissions that demonstrated interdisciplinarity. Some of the topics in the call included: novel user interfaces; information visualization; social aspects of computing; legal issues in computing; structural computing; open hypermedia systems; software engineering; philosophy of computation; collaborative work; workflow processes; object-oriented programming; design patterns; component-based systems; middleware; programming environ-

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 1–5, 2003.

ments; programming languages; operating systems; development environments; agent-based systems; and, information retrieval.

The symposium was structured somewhat differently than many traditional conferences by scheduling sessions intended to foster dialog within small groups of researchers in addition to plenary sessions to stress unifying themes. There was ample opportunity for in-depth discussions to follow up on ideas presented at larger sessions, as well as several informal gatherings to help foster a sense of community in this new area.

During the sessions, much of the discussions revolved around the notions of abstraction, contextualization, and formalism, and the interplay among these concepts. Most speakers used Engelbart's notion of A, B, and C level work as a touchstone for their presentations, tying together diverse topics into a coherent whole.

2 Sessions

There were nine sessions during which papers were presented. Each of these sessions was 75 minutes long.

2.1 Foundations

The "Foundations" session concentrated on laying some basic groundwork for the discussions in the remainder of the symposium. Nürnberg began by describing a vision for metainformatics, and provided several metaphors to which several speakers over the next sessions were able to refer. This provided the symposium attendees an initial (even if constantly revised) common vocabulary. Wiil descibed experiences with the Construct strutcural computing environment, with a special focus towards integration of development ("C-level") tools into the environment itself. The *bootstrapping approach* described by Wiil, in which development tools are client of the system under development, is very reminiscent of Engelbart's bootstrapping approach.

2.2 Knowledge Management

The "Knowledge management" session dealt with various aspects of the knowledge management problem, an area with clearly B- and C-level aspects. Rubart and Wang discussed meta-modeling in the context of knowledge management. They presented a thoughtful analysis of knowledge management from the perspective of Engelbart's levels of work, and identified several specific B- and C-level tasks in the knowledge management context. Tochtermann stressed the organizational and social aspects of knowledge management, especially at the B- and C-levels of work. He argued that personalization is a key concept for B-level knowledge management work, especially with respect to developers of knowledge management software.

2.3 Structural Computing

The "Structural computing" session featured two papers based on work carried out at the University of Patras, Greece. Both papers share the belief that structural computing is inherently "metainformatical" since it explicitly focuses on building tools for tool builders (B- and C-level work). Karousos described how taxonomic reasoning, implemented within a structural comuting environment, could be used to support the A-level task of Web page bookmarking. Tzagarakis described efforts to integrate the BalkaNet project – concerning translation through the use of WordNets, into a structural computing environment. He claimed that the explicit focus on low-level, reusable structure consititutes a ready-made C-level environment.

2.4 Software Engineering 1

The "Software engineering 1" session focused on software engineering as an inherently B-level undertaking, since it constitutes an improvement process of A-level (front-line) work in organizations – namely, generation of systems. Improvements in the software engineering process, in this light, are easily seen to be C-level undertakings. Dalcher argued for a dynamic feedback model of software engineering processes, since static models lead to mismatches between modelled and experienced reality. Whitehead claimed that improvements in modelling notation – specifically, containment modelling notation for content management systems – constitute a C-level improvement.

2.5 Software Engineering 2

The "Software engineering 2" session picked up where the previous session left off. Chen described work to implement (and implicitly model) multithreaded objects more cleanly and correctly, with expected efficiency gains for those software designers engaged in the modelling task. Ekdahl provided a highly theoretical look at programs themselves as reality modelling constructs, and discussed some of the inherent characteristics of this undertaking and our views towards it. His claim is that viewing programs as semantically laden linguistic objects provides leverage.

2.6 Infrastructure

The "Infrastructure" session concentrated on open hypermedia systems, which have long been considered as B-level technology useful for front-line users to engage in various knowledge work. Hicks asks where the expected user community for open hypermedia systems is, since to date, such systems have failed to garner much widespread support among users, despite their apparent advantages over comparable systems. Griffiths and Millard described initial thinking aimed at tackling the difficult problem of versioning in open hypermedia systems. Additions like versioning to such systems might make their adoption more widespread, and thus increase the value of this B-level technology.

2.7 Databases and Data Mining

In the "Databases and data mining" session, the speakers considered problems of data management, capabilities and technologies central to a variety of other, "higher-level" applications discussed at the workshop, such as knowledge management or software engineering. Roantree described a metamodel for federated multimedia databases, arguing that such a metamodel provides users and administrators of such federated systems with a clearer picture of the data owned and how it is structured. Arotaritei described approaches to data mining on the web using the new XLink standard for external structure, claiming that the mining process can be greatly improved in the light of the additional information encoded in this new standard.

2.8 Agents and Multimedia

The "Agents and multimedia" session, as with the previous session, provided a look into the core technologies that enable applications such as knowledge management and software engineering. Reich proposed "EMMO's," or enhanced multimedia meta objects, with the intent on dissociating such multimedia objects from their usage context, enabling repurposing of these objects as appropriate. Zeid, in his paper, considers extensions to well-known modelling methodologies and formalisms to account for agents.

2.9 Position Statements

The "Position statements" session provided a forum for a wide variety of work. Bendix explored additional applications of configuration management principles and technologies, especially social and group applications. Kezmah considered the very topical Web services field, concentrating especially on how these are best viewed from a business perspective. Wagner considered some of the implications of the traditional naming and location problem, with a special emphasis on the human usability of any proposed solution. All of these presentations placed the human in the center of the problem, viewing technologies as tools used for some "higher," front-line purpose, giving us a slightly different view of what B-level work might mean.

3 Wrap-up

A major focus of the symposium was to gather together researchers from disparate areas to share experiences, insights, and best practices through abstraction and generalization of their work. Most attendees agreed that this goal was met. In particular, there were many thought-provoking and extended exchanges among researchers representing the structural computing, software engineering, and knowledge management areas of research.

Preparations for MIS '03 (Santa Cruz, CA, USA) were already underway by the end of MIS '02, with conference venues and chairs for MIS '04 (Salzburg, Austria) and MIS '05 (College Station, TX, USA) selected as well. Information on upcoming events in the Metainformation Symposium Series, as well as information about previous meetings, can always be found at `http://cs.aue.auc.dk/mis/`.

References

1. Engelbart, D. C. 1992. Toward high-performance organizations: a strategic role for groupware. *Proceedings of the GroupWare '92 Conference*, Morgan Kaufmann.

Building Metainformatical Bridges

Peter J. Nürnberg

Department of Computer Science
Aalborg Univeristy Esbjerg
Niels Bohrs Vej 8
DK-6700 Esbjerg, Denmark
pnuern@cs.aue.auc.dk
http://cs.aue.auc.dk/~pnuern/

1 Introduction

In this paper, we first consider Engelbart's notion of "A", "B" and "C" level work, and a broadening of the orginal intent of these terms. We use this new broader definition to characterize the types of improvement processes that computer scientists and informaticists pursue. This allows us to abstract much of informatics work into broad categories, which can lead us to a way of sharing "best practices" among several different fields – even between fields that may appear unrelated.

2 Level of Work

Douglas Engelbart has spoken of "A", "B" and "C" level work – that is, work serving the primary function of an organization, support for this primary work, and support for this support, respectively. For example, imagine a widget factory. The manufacturing of widgets might be the factory's A level work. Work toward widget production process improvements (e.g., better widget assembly equipment) might be an example of the factory's B level work. Finally, effective knowledge management tools for workers concentraring on production improvements (e.g., more effective tools to share production improvement information) might be an example of the factory's C level work. A central tenet of Engelbart's work is that improvements at the C level in an organization contribute more heavily to the overall effectivness of an organization than improvements at the B level, which, likewise, are more "important" than improvements at the A level.

Informatics (and computer science) rarely deal with A level work of organizations. The computer is a "swiss-army knife" tool, applicable in a wide variety of situations, but it remains a tool *to accomplish some other, primary task*. Few organizations work with computers solely for the sake of doing so (excepting certain computer-oriented organizations, of course). Rather, most people use computers to support their organization's primary task.

As such, a computer is, for most people, a B level tool. When we design and implement tools for the computer as B level tool, we are doing B level work.

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 6–8, 2003.

Designing and implementing productivity tools such as spreadsheets or word processors, installing and maintaining network services such as email and web tools, and supporting common tasks through information retrieval services are all examples of B level work in many organizations.

Many other informatics fields deal with C level work, in that they seek to improve the B level work described above. Software engineering and knowledge management are two quite clear examples. Work in these fields is aimed directly at supporting and improving B level work.

Of course, one organization's B level work may be another's A level work. Though originally formulated to describe work within a given organization, we see value in broadening the A-B-C work descriptions. We view the A, B, and C level designations as concerning roles played by a work process. Thus, nearly any process has A, B, and C level aspects. For example, instead of thinking of information retrieval work as B level or C level, we can consider it from a "B angle" or a "C angle", as fits our needs.

3 Bridging the Gaps between Fields

Informatics is often viewed in a very "task-driven" way – fields are related to one another if the products of one field are used by another. Computability theory may influence algorithmic analysis, which may in turn influence the way in which certain information retrieval processes are coded, which in turn may influence the functionality presented in an interface of a word processor, etc. In this way, fields such as computability or algorithm analysis may be seen as "more fundamental" than information retrieval or interface and interaction design. Improvements in algorithm design are likely to have greater impact on a wider variety of computer applications than (non-algorithmic) improvements in information retrieval, since algorithm design underlies more work. In the Engelbart parlance, algorithm design might be seen as C level work, whereas information retrieval might be seen as B level work. (Perhaps in this example, document preparation using word processors might be the relevant A level work.)

However, in our conception of A-B-C labels, in which these labels reflect the roles of the work performed, and not any fundamental aspect of the work itself, these distinctions disappear. Both algorithm design and information retrival may rightly be seen as fitting at any level, since in practice, improvements in either field may (directly or indirectly) facilitate improvements in the other.

This view suggests another grouping of fields. Instead of finding "data-flow chains" of fields (in which products of field X feed proceeses in field Y), we might concentrate on what similarities exist among a set of fields when viewed *at the same level*. For example, what similarities between algorithm design and information retrival can we find when viewing both as A level work? Or as B level work? Or as C level work?

We contend that focusing on the A level of the hierarchy will be less fruitful than at the B or C levels. The differences between algorithm design and information retrieval *as front line tasks* may be too large to overcome. But, viewed

as improvement processes for some other type of work, we may be able to see patterns emerge. For example, whether one's area of speciality is algorithm design or information retrieval, if one is tasked to improve document preparation, one may work toward this goal in similar ways: collecting information about the current state of the document preparation process; determining which parts of the current process are least efficient; or, testing various proposed improvements.

This way of looking at fields and finding similiarities requires us to think in a very abstract way about informatics work. It also urges us to break through "traditional" alignments of fields and find new connections based on these abstractions. We suggest the term "metainformatics" to describe this way of "rising above" traditional field descriptions to find best practices in building and executing improvement processes.

Lessons Learned with the Construct Development Environment

Uffe Kock Wiil

Department of Computer Science and Engineering
Aalborg University Esbjerg
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
ukwiil@cs.aue.auc.dk

Abstract. Construct is a state-of-the-art component-based structural computing environment. The purpose of this paper is to present the most prominent lessons learned with the Construct development environment. In doing so, the paper first sets the context for the work by describing the historical background and rationale behind the work. The next step is to provide an overview of Construct and the development tools that make up the Construct development environment. The paper also enumerates the development experiments from which the empirical data used in this paper are gathered. Then, the paper describes the lessons learned and outlines current and future work targeted at improving the development environment. We believe that many of the lessons learned with the Construct development environment are relevant and applicable to development of other similar component-based distributed software systems.

Keywords: open hypermedia systems, component-based open hypermedia systems, structural computing, software development environments, distributed software systems.

1 Introduction

Open hypermedia systems have been around for more than a decade (e.g., [1,6,7,9,21]). The key characteristics of open hypermedia systems are the provision of external links, or more generally external structures (structures that are managed and stored separately from the documents) and the ability to integrate an open set of desktop applications to make use of the provided structure support.

A recent trend in open hypermedia work is to develop systems as sets of services wrapped in separate service components. These component-based open hypermedia systems are more flexible, in particular with respect to developing and adding new services to the open hypermedia environment. Another recent trend is to provide different types of structure services (in addition to the standard navigational structure service providing links) within the same environment. The term structural computing was coined to describe such environments [8].

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 9–17, 2003.

In 1999 a workshop series was initiated to discuss important issues and advance the thinking about structural computing [10,13,14].

Construct [4,16,17] is a state-of-the-art component-based structural computing environment developed as the common code base successor of the DHM [6], HOSS [9], and HyperDisco [21] open hypermedia systems. Development of new functionality in such complex distributed software systems requires expert knowledge of the target system. The goals with the Construct development environment is to lower the entry-barrier for developers by basing the environment on components that use well-defined design patterns and templates. The development tools take care of many of the complex design decisions and coding tasks. The developer does not need to know details of synchronization, multi-threaded programming, or remote method invocation. New service interfaces are defined in high-level specifications with the help of dedicated development tools. Only a moderate level of programming expertise is required to develop new service components in Construct.

The purpose of this paper is to present the most prominent lessons learned with the Construct development environment. We believe that many of these lessons are relevant and applicable to the development of other similar component-based distributed software systems.

Section 2 presents the background for the present work on development environments. Section 3 provides an overview of Construct and the development tools that make up the Construct development environment, and enumerates the development experiments from which the empirical data used in this paper are gathered. Then, in Sect. 4, we describe the lessons learned and, in Sect. 5, we outline current and future work targeted at improving the development environment. Section 6 concludes the paper.

2 Why Development Tools?

Many of the ideas behind the present work on development environments were conceived in the Hyperform [20] and HOSS [9] projects as well as the work in the Open Hypermedia System Working Group (OHSWG) [11].

Hyperform (from the early 1990's) is both a hypermedia system development environment and a hypermedia system execution environment. The focus was on rapid prototyping of hypermedia services such as system architectures, data models, and entire hyperbase configurations. These hypermedia services were kept in libraries to enable reuse. Hyperform provides a set of editors (development tools) that allow hypermedia services to be specified and installed in the environment at runtime.

HOSS (from the mid 1990's) focused on providing development tools that ease the task of generating new hypermedia services. Examples of development tools in HOSS are the Protocol Definition Compiler and the Generalized Process Template. Experiences from HOSS indicate that about 90% of the code base of a typical hypermedia system can be auto-generated based on high-level service specifications.

The OHSWG started its work on open hypermedia interoperability and standards in March 1996 [22]. The first few years of OHSWG effort focused on defining standards that allowed for interoperability between existing open hypermedia systems. The first standard that was defined covered the navigational hypermedia domain (called OHP-Nav) [5,15].

It is a major goal of the Construct project to be fully OHSWG standards compliant. For example, Construct provides a complete implementation of the OHSWG navigational interface standard (OHP-Nav). One important lesson from this work is that a standard goes through several iterations before it is ready to use. In every single one of these iterations, parts of the interface are updated. The result is that all systems that wish to comply with the standard must be updated several times as the standard evolves.

Our background with the Hyperform and HOSS systems led us to believe that a development environment with a set of basic development tools can help reduce the effort to develop new standards and the effort for system developers to comply with these standards when developing service components.

3 Construct Structural Computing Environment

Figure 1 provides a conceptual overview of Construct – for additional details see [16] and [17]. The environment consists of three categories of Construct services that interoperate with existing desktop applications (e.g., Web browsers and Microsoft Office applications). These desktop applications manipulate data located in different data stores (e.g., file systems or Web servers). Construct development tools assist developers in creating new Construct services. Editor and compiler tools allow developers to specify and generate new Construct service components. Infrastructure services allow these components to coexist in the component-based environment. Different structure services serve hypermedia domains (e.g., links, classification structures, metadata, annotations, etc.) to existing desktop applications. Structure services rely on structure stores.

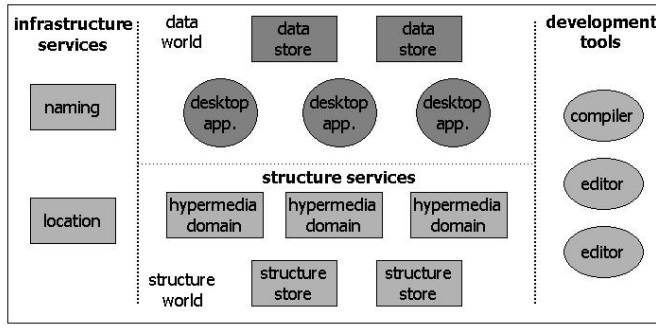


Fig. 1. A conceptual overview of the Construct structural computing environment

Section 3.1 describes the development tools and how they are used. Section 3.2 enumerates the tools and services that have been developed using the Construct development environment.

3.1 Construct Development Environment

This section provides a brief overview of the Construct development tools. Additional details can be found in [3,18,19].

Figure 2 provides an overview of the development tools available in the Construct environment and the first few steps in the process in which they are deployed. The circle at the top represents the service developer using the development tools (represented by screen shots). The ovals at the bottom represent products at different stages in the development process.

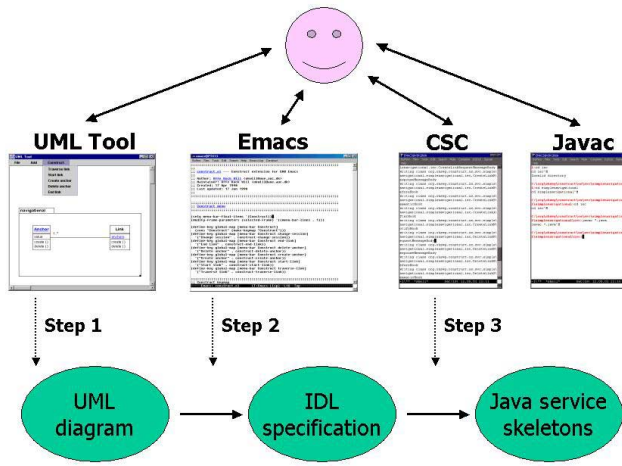


Fig. 2. The first few steps in the development process with the Construct development tools. The remaining steps in the development process are the classical tasks of coding and compilation. Emacs can also be used for those steps (from [18])

The overall steps in the development of new services in the Construct structural computing environment are:

1. UML Tool [3] is used to create a UML diagram specifying the interface of the required service. When the diagram is ready, UML Tool can generate an IDL specification from the UML diagram. UML Tool is a client of the Construct navigational structure service. Both class names and field names can serve as endpoints of links. This allows the developer to create links (associations) between (parts of) service specifications and other documents (i.e., design rationale). See [18] for a detailed description of UML Tool and a detailed development scenario of its use in service component specification.

2. Emacs is used to display and edit IDL specifications. If the developer prefers to write an IDL specification of the service interface directly (instead of starting with a UML diagram), then this step is the first step in the development process. Emacs is a client of the Construct navigational structure service as well as the Construct metadata service. Arbitrary text strings can serve as endpoints of links. Each file in Emacs can have a metadata record attached. A metadata record consists of arbitrary key/value pairs such as author name, date created, comment, etc.
3. The Construct Service Compiler (CSC) transforms the service interface specification in IDL into a Java skeleton service component. The CSC runs inside a shell tool (e.g., Emacs). See [19] for a detailed description of how the CSC works and what specific types of output it generates.
4. Emacs is used to edit the Java source code in the skeleton service. The developer creates the semantic parts (method bodies) of the operations defined in the service interface. When the semantic parts are added to the skeleton service, the service is fully developed.
5. The Java source code for the skeleton service is compiled into a Construct service component that automatically is operational in the Construct structural computing environment. The Java compiler (Javac) runs inside Emacs in its own shell.

As illustrated, the developer can decide to work at different levels of abstraction when specifying the service interface. Services interfaces can be specified graphically as UML diagrams in UML Tool or textually as IDL specifications in a text editor (e.g., Emacs). The use of the development tools makes the coding step (Step 4 above) easier. The tools handle many difficult design decisions and generate much of the complex code based on the interface specifications.

3.2 Use Experiences with the Construct Development Environment

The Construct development tools have been used to develop more than 15 services. The fully developed services include a structure storage service, a navigational service and wrapper services[1] for integrating applications (Netscape, Emacs, and UML Tool), a metadata service and wrapper services for integrating applications (Netscape and Emacs), and a set of cooperation services. Several structure services are currently under refinement including a spatial, a taxonomic, and a data mining service. See [16] and [17] for details about these services.

4 Lessons Learned

This section presents the most prominent lessons learned from using the Construct development tools. Some of these experiences have been reported earlier in [18] and [19]. The lessons are grouped into four overall categories:

[1] A wrapper service is a service that helps integrate a particular desktop application, such as Netscape or Microsoft Word. A wrapper service mediates the communication between a structure service (e.g., a link service) and the application.

1. **Reduced development effort and complexity lowers the entry barrier.** The development environment reduces the effort involved in developing and maintaining Construct service components – both with respect to complexity and time. All developed services automatically become part of the Construct environment, which can operate in a massively distributed environment. The "automation" of complex parts of the development of service components (through the use of templates and design patterns) is an important factor in lowering the entry barrier for service developers. The level of expertise needed to develop service components for the Construct environment is considerably lower than the level of expertise needed to build similar service components from scratch without the development tools [19].
2. **Service development adds to the library of service components.** The library of service components grows each time a new service is added to the Construct environment. This library is extremely helpful for services developers. Software reuse at the component level can save considerable time when developing new services. This applies to both high-level specifications of components in UML or IDL and to implementations of service specifications in Java. Component specifications can be reused when developing services that are some kind of a variation (e.g., a specialization or a generalization) of an existing service. Implemented service components that provide a useful (general) functionality can be deployed directly in a new setting and need not be developed again.
3. **Flexible and structured working styles.** The development environment enforces a structured way of working. It is very simple: the services must be specified in UML or IDL before the implementation in Java can be initiated. The development can be done all at once (in one development cycle) or in a number of iterations. The latter supports a (rapid) prototyping approach to service development, where the service specification (and implementation) is refined in a number of steps. Also, debugging can now partially take place at the service specification level rather than at the source code level. Many errors can now be caught either when the UML diagram is constructed or when the IDL specification is parsed in the Construct Service Compiler.
4. **Coping with new standards and technologies.** The Construct Service Compiler is created in a modular way, which allows new technologies to be incorporated and new outputs to be generated in a straightforward manner. If a new communication standard needs to be adopted in the environment (i.e., to make the components CORBA compliant), only the modules handling the communication in the component framework needs to be modified. If a new form of output is requested, say software agents, new output modules generating software agents based on UML or IDL specifications can be added to the compiler.

5 Ongoing and Future Work

A new service generator, the Construct Service Generator (CSG), is currently under development. Based on the experiences with the Construct Service Com-

piler (CSC), the CSG constitutes a significant improvement over the CSC. A number of improvements are made and new features are added including:

1. The use of Java RMI as communication standard between components is added. The CSC provides TCP/IP and shared memory as means to communicate between components.
2. The CSG generates build files for "Ant" (`http://jakarta.apache.org/ant`). "Ant" is a Java based build-tool similar to "Make".
3. The Java code generated by the CSG has a much simpler class structure than the Java code generated by the CSC. This makes it easier for service developers to comprehend and use the generated source code.
4. The CSG generates two sets of API's for the service components – a new CORBA compliant API and an API that is backward compatible with the CSC.

So far our effort has resulted in a set of stand-alone tools that can be deployed in different phases of service development. In our future work we also plan to examine ways to achieve a smoother integration of the development tools by deploying control integration techniques. A simple example of control integration is to have UML Tool initiate the CSC with the generated IDL specification. This will allow developers to generate the service component skeleton from the UML diagram in one step. Having the CSG generate Ant build files is also a step in this direction.

6 Conclusions

The paper has presented the most prominent lessons learned from the development and use of the Construct development environment. From a general software development point of view it is probably not surprising that software development tools actually work. This is in fact also our primary experience with the Construct development environment, which is targeted at developing structural computing services. However, we believe that many of the lessons that we have learned can be generalized and the tools that we have developed could be useful in the development of other types of component-based distributed software systems.

The Construct development environments share common features and goals with commercial development environments such as Rational Rose [12] and the ArgoUML [2] open source project. We have found two noticeable differences between our approach and approaches like Rational Rose. First of all, the Construct software is publicly available free of charge. Secondly, we have taken an open systems approach to the development environment, which aims at re-using and integrating with existing applications, tools, and services in the computing environment. Only the second difference applies to approaches like ArgoUML – since it is also publicly available free of charge.

Acknowledgments

Several people have been involved in the development of Construct. In this context, we wish to acknowledge people contributing to the development environment. Peter Nürnberg developed the Construct Service Compiler (CSC) and is currently working on its successor the Construct Service Generator (CSG). Stéphane Blondin and Jérôme Fahler, two Socrates exchange students from the University of Bretagne Occidentale in Brest, France, implemented the UML Tool during a three-month visit during the summer of 2000 at the Department of Computer Science and Engineering, Aalborg University Esbjerg.

References

1. Anderson, K. M., Taylor, R., and Whitehead, E. J. 1994. Chimera: Hypertext for heterogeneous software environments. *Proceedings of the 1994 ACM Hypertext Conference* (Edinburgh, Scotland, Sep), ACM Press, 94-107.
2. ArgoUML. 2002. `http://argouml.tigris.org`.
3. Blondin, S., Fahler, J., Wiil, U. K., and Nürnberg, P. J. 2000. UML Tool: High-level specification of Construct services. Technical Report CSE-00-01, Department of Computer Science and Engineering, Aalborg University Esbjerg.
4. Construct. 2002. `http://www.cs.aue.auc.dk/construct`.
5. Davis, H. C., Millard, D. E., Reich, S., Bouvin, N. O., Grønbæk, K., Nürnberg, P. J., Sloth, L., Wiil, U. K., and Anderson, K. M. 1999. Interoperability between hypermedia systems: The standardisation work of the OHSWG. *Proceedings of the 1999 Hypertext Conference* (Darmstadt, Germany, Feb), 201-202. ACM Press.
6. Grønbæk, K., and Trigg, R. 1999. *From Web to Workplace – Designing Open Hypermedia Systems*. MIT Press.
7. Hall, W., Davis, H., and Hutchings, G. 1996. *Rethinking Hypermedia – The Microcosm Approach*. Kluwer Academic Publishers.
8. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As we should have thought. *Proceedings of the 1997 ACM Hypertext Conference* (Southampton, UK, Apr), ACM Press, 96-101.
9. Nürnberg, P. J., Leggett, J. J., Schneider, E. R., and Schnase, J. L. 1996. HOSS: A new paradigm for computing. *Proceedings of the 1996 ACM Hypertext Conference* (Washington, DC, Mar), ACM Press, 194-202.
10. Nürnberg, P. J. (Ed.) 1999. *Proceedings of the First Workshop on Structural Computing*. Technical Report CS-99-04, Department of Computer Science, Aalborg University Esbjerg, Denmark.
11. Open Hypermedia System Working Group. 2002. `http://www.cs.aue.auc.dk/ohs`.
12. Rational Rose. 2002. `http://www.rational.com/rose`.
13. Reich, S., and Anderson, K. M. (Eds.) 2000. *Open Hypermedia Systems and Structural Computing*. Lecture Notes in Computer Science vol. 1903, Springer Verlag.
14. Reich, S., Tzagarakis, M., and De Bra, P. M. E. (Eds.) 2001. *Hypermedia: Openness, Structural Awareness, and Adaptivity*. Lecture Notes in Computer Science vol. 2266, Springer Verlag.
15. Reich, S., Wiil, U. K., Nürnberg, P. J., Davis, H. C., Grønbæk, K., Anderson, K. M., Millard, D. E., and Haake, J. M. 1999. Addressing interoperability in open hypermedia: The design of the open hypermedia protocol. *The New Review of Hypermedia and Multimedia 5*, 207-248.

16. Wiil, U. K., Hicks, D. L., and Nürnberg, P. J. 2001. Multiple open services: A new approach to service provision in open hypermedia systems *Proceedings of the 2001 ACM Hypertext Conference*, (Århus, Denmark, Aug), ACM Press, 83-92.
17. Wiil, U. K., and Hicks, D. L. 2001. Tools and services for knowledge discovery, management and structuring in digital libraries. *Proceedings of the Eighth ISPE International Conference on Concurrent Engineering: Research and Applications* (Anaheim, CA, Jul), 580-589.
18. Wiil, U. K. 2001. Development tools in component-based structural computing environments. In [14], 82-93.
19. Wiil, U. K. 2000. Using the Construct development environment to generate a file-based hypermedia storage service. In [13], 147-159.
20. Wiil, U. K., and Leggett, J. J. 1997. Hyperform: A hypermedia system development environment. *ACM Transactions on Information Systems 15*(1), 1-31.
21. Wiil, U. K., and Leggett, J. J. 1997. Workspaces: The HyperDisco approach to Internet distribution. *Proceedings of the 1997 ACM Hypertext Conference* (Southampton, UK, Apr), ACM Press, 13-23.
22. Wiil, U. K. (Ed.) 1997. Open hypermedia: Systems, interoperability and standards. *Journal of Digital Information 1*(2).

A Meta-modeling Environment for Cooperative Knowledge Management

Jessica Rubart[1], Weigang Wang[1], and Jörg M. Haake[2]

[1] Fraunhofer Institute for Integrated Publication and Information Systems (IPSI)
Dolivostrasse 15 64293 Darmstadt, Germany
{rubart, wwang}@ipsi.fhg.de
[2] FernUniversität Hagen, Computer Science VI - Distributed Systems
Informatikzentrum, Universitätsstrasse 1
58084 Hagen, Germany
joerg.haake@fernuni-hagen.de

Abstract. Knowledge management (KM) systems usually focus on specific tasks. In this paper, we identify B- and C-level improvement activities for cooperative KM based on Engelbart's ABC model of organizational improvement. We present a cooperative meta-modeling environment, which is based on cooperative hypermedia. It supports the configuration of domain-specific knowledge schemas and their ad-hoc adaptation with special consideration of behaviors.

1 Introduction

Knowledge management (KM) is very important for enterprises in order to deal with huge amounts of information produced and manipulated by e.g. projects. Efficient KM can for example reduce costs, shorten project execution time, increase product quality and support learning. Techniques to organize and capture knowledge are crucial. KM deals with the creation, dissemination, and utilization of knowledge. According to Engelbart [4] groupware plays an important role in supporting knowledge capabilities. More and more work needs to be done concurrently in teams and across organizational boundaries. Such work can be seen as knowledge processes, which are cooperative, dynamic and continuous, and manipulate an evolving knowledge base. In [14] cooperation is even presented as one face of KM. We use the term *cooperative knowledge management* in order to include cooperation explicitly.

KM systems usually focus on specific tasks, such as searching with search engines, organizing knowledge bases on the web [12] or capturing and exploiting tacit knowledge from speech [3]. However, there is no support for knowledge processes and evolving knowledge schemas that can improve KM activities. Referring to Engelbart's ABC model of organizational improvement the work presented in this paper focuses on B-level activities for improving KM, such as providing support for cooperative process modeling and execution. In addition, it focuses on C-level activities for improving B-level activities and in turn A-level

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 18–28, 2003.

activities. Examples of such C-level activities include tailorability, extensibility and support for the execution of structure.

The next section presents a problem analysis and identifies requirements for an environment, which supports B- and C-level activities for cooperative KM. Then, our approach addressing these requirements including tools and an example scenario related to software engineering are described. Afterwards we compare our approach to related work. The paper ends with conclusions and plans for future work.

2 Problem Analysis

According to Engelbart's ABC model of organizational improvement [4] A-level activities represent core business activities. What are core activities of KM? Ricardo points out four major faces of KM [14]. These are the search engine view, the document-management view, the data-mining-tool view, and the social nexus view. The latter focuses on cooperation. Thus, in the KM area A-level activities include searching for documents or special information, organizing documents with version and security control, extracting useful knowledge from document bodies, and creating online virtual meeting spaces for cooperation. In addition, Ricardo points out that work in hypertext research can enrich KM systems, such as intelligent linking of documents to support epistemic exploration of search results.

For improving such A-level activities we think that supporting knowledge processes is crucial. Thus, activities related to process support are important B-level activities. Process support improves the organization's ability to perform A-level work. The different activities can get harmonized in a process description, which serves as shared information. This explicitly answers questions, such as when to hold a meeting or work on a document, whether existing processes can be reused, or how to resolve a resource conflict. This requires support for:

- **R1.** the definition of work processes and the assignment of resources to processes, such as assigning documents and users to tasks;
- **R2.** cooperative access to and manipulation of shared work processes;
- **R3.** execution of shared work processes, e.g. providing users with to-do lists, providing access to required resources and services, changing the states of tasks; and,
- **R4.** learning of process knowledge, e.g. capturing best practice examples.

For improving these B-level activities it is important to improve cooperative modeling and execution capabilities. Domain-specific modeling schemas support the different languages of different groups of end-users, e.g. a "task" might need to be called an "exercise" in a learning setting. These semantic types need to be applicable on processes as well as on all related information objects, such as documents or user representations/objects. In addition, the process structure can be enriched with new document types, such as a brainstorming for special

cooperation support. For this, extensibility mechanisms are required. Finally, an open set of execution services working on the whole process and document structure can improve B-level activities since execution then exceeds work process support. Extensibility and tailorability support for modeling and execution capabilities implies the provision of a clear model for expressing extensions and a set of tools [10] for implementing them. It needs to be "easy" to extend and tailor systems compared to just modifiability and openness. Thus, the C-level activities need tool support for:

- **R5.** cooperative definition of semantic types, i.e. end-users as well as application developers need to be able to tailor the modeling language to specific domains;
- **R6.** cooperative assignment of semantic types to object types, i.e. end-users and application developers identify required object types (related to knowledge units as well as relationships between these) and label them according to the domain language;
- **R7.** extensibility of object types, i.e. support for registering new object types so that these can be integrated in the system;
- **R8.** cooperative tailoring of the domain-specific schema, i.e. end-users and application developers can specify, which semantic types of relationships are allowed between which semantic types of knowledge units; and,
- **R9.** registering execution services working on domain-specific schemas in order to provide an extensible set of these.

Figure 1 shows the identified capabilities based on Engelbart's ABC model. The next section describes our approach for addressing the identified requirements.

Figure 1 shows levels A, B, and C presented as in [4] and including the identified capabilities. The B- and C-level capabilities we have identified are probably not the only capabilities one can identify on these levels, but important needs from our experiences. The arrows in the figure indicate improvement. According to Engelbart there are also B- and C-level activities that can improve activities of the same level (illustrated by the loops). The lower circle in the figure represents Engelbart's "point of greatest leverage" and thus refers to output of activities that increases C, B, and A capabilities. Our identified capabilities fit nicely into this model, e.g. registering a new object type (C-level) results in improved tailorability capabilities, which are on the C-level, too. This improves assignment possibilities of objects (documents) to processes, which is on the B-level. This, in turn, improves A activities, such as working on a document of the new type.

3 Interactive Configuration of a Cooperative Hypermedia System

Our approach is based on experiences from our earlier work on CHIPS [18,19] and designed as an extension of XCHIPS [15,20]. This extension *XCHIPS4KM*

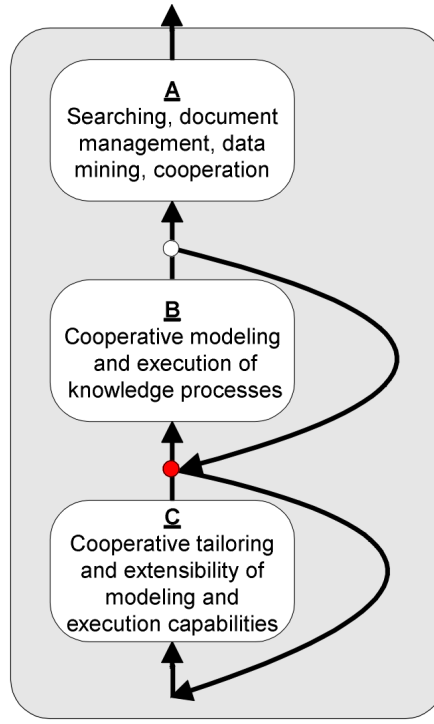


Fig. 1. A-, B-, and C-levels for Cooperative Knowledge Management

includes several tools for supporting C-level work. Firstly, we describe how XCHIPS supports knowledge processes as B-level work. Then, we present the extensions to XCHIPS for supporting C-level work.

3.1 Supporting Knowledge Processes

We model knowledge processes using cooperative hypermedia with process support [18]. This supports users in creating both the knowledge structure and the process description (as required by R1). Hypermedia-based structure is well suited to represent knowledge by using typed links and typed nodes [8]. Our model uses special node and links types for representing process structures explicitly. In addition, every hypermedia object is also a shared object with a unique identifier, shared attributes and the possibility to make it persistent [15]. Concurrency control and change propagation mechanisms are applied on shared hypermedia objects. This supports users in accessing shared and persistent hypermedia objects, either synchronously or asynchronously, and interact with them through tools (as required by R2). Our approach provides cooperation-aware visual modeling capabilities to create explicit process models embedded in hypermedia-based knowledge structures.

Our explicit support for process structures enables execution of the modeled processes (as required by R3), e.g. setting a task's state to completed sets the state of following tasks to enabled. By using queries work process support, such as to-do lists, is provided.

Learning of process knowledge means to learn what to do and to learn how to do it [20]. In our approach, both the learning process and the process to be learned can be modeled and executed cooperatively. Learning (as required by R4) is supported by executing processes for simulation purposes, by experts giving guided tours through process descriptions and best practice examples, and by cooperative role-play for practicing the tasks.

Best practice examples can be seen as special process patterns. A pattern names a design structure as a solution to a problem in a particular context and thus increases our vocabulary [1,7]. It's like a template that one can apply in different situations. In our approach, there are reusable process-centered knowledge structures. In general, our approach provides the possibility to use any node, link or composite instance as a pattern. It can be cloned and adapted.

The process-centered knowledge structures can guide A-level work. Documents and tools can be integrated into the structure and used when appropriate. In addition to the cooperative modeling and execution capabilities explicit communication channels can be used.

3.2 Tailorability and Extensibility of Modeling and Execution Capabilities

In order to support visual modeling based on domain-specific modeling schemas it is important to configure the "language" of the end-users. For this our meta-modeling environment provides the possibility to specify semantic types. Basically, a semantic type includes a name and an icon. It is comparable to a UML stereotype [11] as it represents a usage distinction of an existing model element. Semantic types capture application domain concepts and relationships and constraints can be defined on them. In the following subsections, we firstly present our hypermedia-based meta-modeling technology, and then describe tools and examples of the meta-modeling environment.

The Hypermedia-Based Meta-modeling Technology. Graphical hypermedia representation is a kind of graph-based visual modeling language. However, our hypermedia visual language differs from many other visual languages in that it has pre-defined, tailorable and extensible computational semantics (i.e., behaviors).

Graphical Representation of Semantic Types. In a graphical hypermedia representation, nodes and links are represented by graphic elements. Normally, nodes are represented using shapes containing an image and/or a text label indicating their type. Lines in different styles are used as representations of explicit links between nodes. A name label attached to the line can also indicate a semantic

type of a link. For one node there can be multiple graphical representations or views. The node view in a directed graph structure can be seen as an iconified (or closed) view of the node. The visual presentation of the content of a node can be seen as an opened view (or content view) of the node. Such a content view can be shown in the area of the closed view, or in another window.

The Behaviors of Semantic Types. The behavior of a node or a link is encapsulated in its type definition and implemented in its object class. The operations available on a node view can be an aggregation of the menu operations defined in the node view object class and the node data object class. The interaction behavior of a node content is reflected in a content view class and the classes for the individual objects contained in the node. When a node contains other nodes and links, the behaviors of its content is the behavior of a hypermedia structure. Similarly, the behavior of a link can be defined in the link view object class and the link data class.

The Behavior Extensibility and Tailorability. In our system, registering and loading new object classes for node views, node data models, and node contents can be used to implement extensible behavior. However, using such a class-based approach, the difficult problem is how to add or change behaviors of existing objects (instances). Our approach to this problem is to use attribute-value pairs to represent properties of an object type. In such an attribute list, the key is the name (string) of an attribute while the value part is a placeholder for an object. In addition to a reference to an object instance, this placeholder also has an attribute for the object type it can hold. An object at client side (i.e., on each cooperative user's machine) is the replica (remote representation) of "the same" object at the server side. The server maintains the persistency of objects. Whenever an object is transferred from the server side to the client side (as serialized data), a replica is created by using its class byte code and the serialized data. The class byte code initially comes from the server, and could be updated when it is changed at the server side. In this way, behaviors can be added or modified by adding or changing methods of the class, recompiling, and registering the class (byte code file) with the server. A translator that converts an old attribute list to a new attribute list handles the property changes.

Tools and Examples. Figure 2 shows our semantic type editor, which supports cooperative definition of semantic types (as required by R5).

Three users are working on the semantic types for a software development project. They have specified types for development processes, such as *Task* and *Milestone* as well as types specifying the artifacts of a software system, such as *Class Diagram*. Carol is representing the application developers who are responsible for C-level support while Alice and Bob are representing the end-users – Alice is a project manager and Bob is a software developer.

Next they assign the semantic types to object types. Figure 3 shows them assigning knowledge unit classes to semantic types by using the menu bar editor.

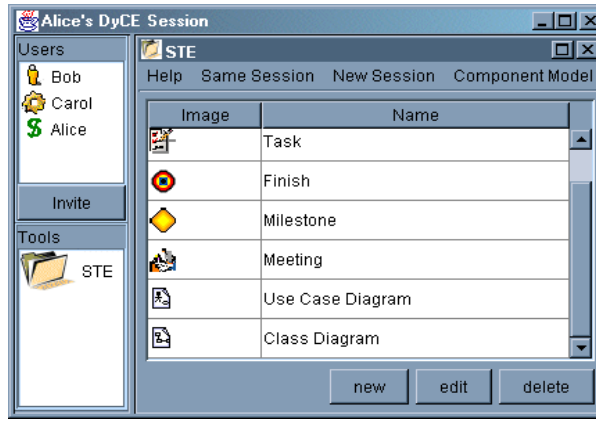


Fig. 2. Semantic Type Editor

It provides cooperation-aware assignment possibilities (as required by R6). The tool provides awareness of who is currently editing assignments. It is used to configure the visual knowledge editor since each assignment specifies a menu item. By using these menu items in the visual knowledge editor end-users can create objects of the specified object types and the knowledge editor visualizes them by using the assigned semantic type. Similarly, XCHIPS4KM offers tool support in order to assign relationship classes to semantic types. When creating a relationship between two knowledge units with the knowledge editor users can select between available semantic relationship types.

If a needed object type is not available, the application developers need to provide it since they are responsible for supporting C-level work. Thus, it makes sense that at least one representative of them joins the cooperative assignment session. This person knows the available object types, can discuss the needs of the end-users with them, and can write down identified requirements. New object types can be registered and integrated into the system as soon as they become available (as required by R7). They can be loaded at run-time without forcing the users to shut down their system similar to groupware components that allow users to access and manipulate shared data objects cooperatively or individually [15,17].

A special schema editor supports the cooperative tailoring of the domain-specific schema (as required by R8). This is based on the semantic types. Figure 4 shows the schema editor and the knowledge editor using the defined schema. The schema editor uses a table-based user interface – similar to the other editors. This is due to the fact that tables are easy to understand for doing assignments. The knowledge editor shows on the left the defined menu bar. Alice, Bob and Carol are currently modeling a structure based on the defined schema. We can see Carol's session window and Alice's telecursor. The schema can be changed at any time.

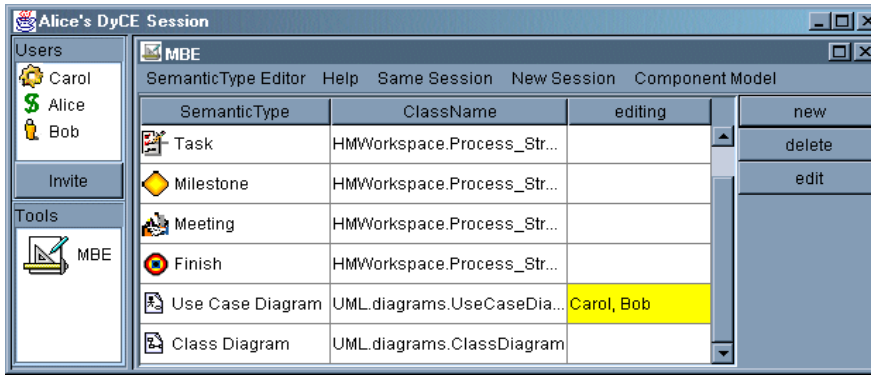


Fig. 3. Menu Bar Editor

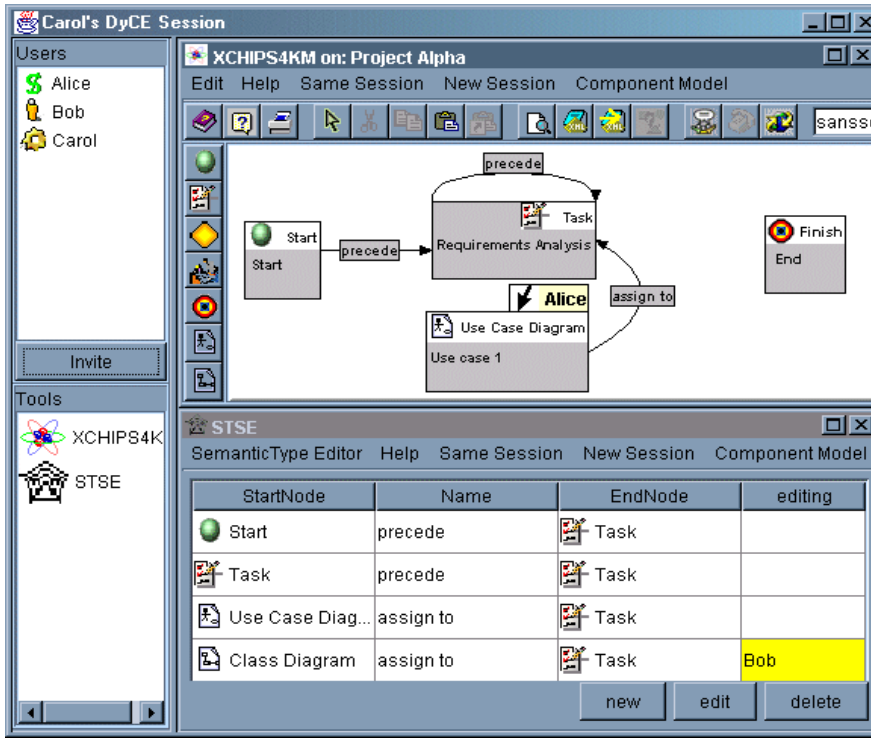


Fig. 4. Schema- and Knowledge Editors

The classes that are assigned to semantic types of nodes (knowledge units) specify the menu items in the popup menus of the nodes and thus the functionality available. Dependent on the type of the underlying object, registered groupware components or external applications can be invoked on these objects,

such as a cooperative use case editor on the use case diagram. Similarly, the classes that are assigned to semantic types of relationships specify the functionality available on these links.

Once a structure has been modeled it can be executed – either partly or as a whole. There is already work process support provided by the B-level capabilities based on a predefined process-centered schema. A different execution service can for example provide a next generation workflow engine that supports the initialization and activation of synchronous cooperation. In order to provide extensibility regarding the execution of structure we are working on an extensible set of execution services (as required by R9). Such services work on special predefined schemata. An open set of execution services is crucial from the structural computing point of view [9]. In order to provide extensibility we are working on registration services for these and an open protocol based on web standards, such as XML. We think that those services can be integrated into component-based open hypermedia systems by using the concept of open service provision [21] and by adding a separate layer for execution services on top of the structure services layer as discussed in [16].

Implementation. The implementation of our meta-modeling environment is based on DyCE [17]. DyCE is a Java-based framework that supports the development of groupware components. Groupware components allow groups of users to collaboratively view and modify shared data objects. Java's classloader abilities are utilized in order to load components and shared data types at runtime. DyCE uses a replicated architecture for shared data objects and provides transactional support for access to and modification of these. Each of our editors is designed as a separate groupware component working on specific shared data. Both – XCHIPS's hypermedia data model as well as XCHIPS4KM's configuration model – are implemented as shared data. We are using an extended DyCE server that includes a special-purpose web server, which provides services for invoking the different components on specified structure.

4 Comparison to Related Work

We have extended our previous work on CHIPS [18,19] and XCHIPS [15,20] by providing several cooperative editors that allow the configuration of a cooperative hypermedia system so that it can be used for different domains and thus as a general cooperative KM system. The example-based schema editor of CHIPS does support schema definition based on semantic types, but does not consider new classes in order to enrich the functionality of the cooperative hypermedia system.

Workflow management systems (WFMS), such as [2], provide work execution support as it is needed for our identified B-level capabilities. However, WFMS usually don't support teams in negotiating joint work processes, in facilitating synchronous access to shared information resources, and in providing immediate interactive feedback and awareness.

Knowledge-centered systems usually provide A-level capabilities, such as support for searching, document management, data mining or cooperation, such as Eidetica [5], Tempus Fugit [6] or FOCI [12]. However, our identified B- and C-level capabilities are not supported.

Component-based open hypermedia systems, such as CAOS [13] or Construct [21], support an open set of structures and thus are very useful for expressing any kind of knowledge. However, they don't provide a cooperative meta-modeling environment for configuring domain-specific knowledge schemas and their ad-hoc adaptation.

5 Conclusions and Future Work

We have identified B- and C-level improvement capabilities for cooperative KM based on Engelbart's ABC model of organizational improvement. Then we have presented how our approach, which is based on cooperative hypermedia and focuses on cooperative meta-modeling, supports the identified requirements. The main technical contributions of our approach are extensibility and tailorability of behaviors of a cooperative hypermedia system, which are used to support B- and C-level activities for improving cooperative KM.

At this time, the first version of our cooperative meta-modeling environment is finished. We are using it in the context of the EU project EXTERNAL, which aims at supporting Extended Enterprises in working on joint projects. Initial experiences suggest that the approach helps our users to adapt the cooperative hypermedia system to special knowledge schemas, such as for progress reports, and make use of the work process support.

In our future work, we will carry out further evaluations of the approach. In addition, we intend to integrate our concepts in component-based open hypermedia systems and try to define some standards for meta-modeling and execution of structure.

Acknowledgments

This work was partially supported by the EU project EXTERNAL, which is funded by the CEC (IST 1999-10091). Thanks also go to Christian Meffert for his implementation support and the anonymous reviewers for their detailed comments. Finally, we would like to thank all attendees of the Metainformatics Symposium for valuable discussions.

References

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. 1977. *A Pattern Language*, Oxford University Press.
2. Ames, C. K., Burleigh, S. C., and Mitchell, S. J. 1997. A Web Based Enterprise Workflow System. *Proceedings of Group'97*, ACM Press.

3. Brown, E., Srinivasan, S., Coden, A., Ponceleon, D., Cooper, J., Amir, A., and Pieper, J. 2001. Towards speech as a knowledge resource. *Proceedings of CIKM'01*, ACM Press.
4. Engelbart, D. C. 1992. Toward high-performance organizations: a strategic role for groupware. *Proceedings of the GroupWare '92 Conference*, Morgan Kaufmann.
5. Eidetica. `http://www.eidetica.com`
6. Ford, D. A., Ruvolo, J., Edlund, S., Myllymaki, J., Kaufman, J., Jackson, J., and Gerlach, M. 2001. Tempus Fugit: a system for making semantic connections. *Proceedings of CIKM'01*, ACM Press.
7. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
8. Nanard, J., and Nanard, M. 1991. Using structured types to incorporate knowledge in hypertext. *Proceedings of Hypertext '91*, ACM Press, 329-343.
9. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As we should have thought. *Proceedings of Hypertext '97*, ACM Press, 96-101.
10. Nürnberg, P. J. 2001. Extensibility in component-based open hypermedia systems. *Journal of Network and Computer Applications 24*, Academic Press, 19-38.
11. OMG 1999. OMG Unified Modeling Language Specification V1.3, `http://www.rational.com/uml/`.
12. Ong, H.-L., Tan, A.-H., Ng, J., Pan, H., and Li, Q.-X. 2001. FOCI: Flexible Organizer for Competitive Intelligence. *Proceedings of CIKM'01*, ACM Press.
13. Reinert, O., Bucka-Lassen, D., Pederson, C. A., Nürnberg, P. J. 1999. CAOS: A collaborative and open spatial structure service component with incremental spatial parsing. *Proceedings of Hypertext '99* ACM Press, 49-50.
14. Ricardo, F. J. 2001. Hypertext and knowledge management. *Proceedings of Hypertext '01*, ACM Press, 217-226.
15. Rubart, J., Haake, J. M., Tietze, D. A. and Wang, W. 2001. Organizing shared enterprise workspaces using component-based cooperative hypermedia. *Proceedings of Hypertext '01*, ACM Press, 73-82.
16. Rubart, J., Wang, W., and Haake, J. M. 2002. Arguments for open structure execution services. *Proceedings of the Open Hypermedia Systems Workshop '02*, FernUniversität Hagen, 295 - 8/2002, 45-51.
17. Tietze, D. A. 2001. A framework for developing component-based co-operative applications. GMD Research Series No. 7/2001, ISBN: 3-88457-390-X.
18. Wang, W., and Haake, J. M. 1998. Flexible coordination with cooperative hypermedia. *Proceedings of Hypertext '98*, ACM Press, 245-255.
19. Wang, W., and Haake, J. M. 2000. Tailoring groupware: the cooperative hypermedia approach. *Computer Supported Cooperative Work: The Journal of Collaborative Computing 9*(1), Kluwer.
20. Wang, W., Haake, J. M., Rubart, J., and Tietze, D. A. 2000. Hypermedia-based support for cooperative learning of process knowledge. *Journal of Network and Computer Applications 23*, Academic Press, 357-379.
21. Wiil, U. K., Hicks, D. L., and Nürnberg, P. J. 2001. Multiple open services: a new approach to service provision in open hypermedia systems. *Proceedings of Hypertext '01*, ACM Press, 83-92.

Personalization in Knowledge Management

Klaus Tochtermann

Know-Center, Inffeldgasse 16c, 8010 Graz, Austria
ktochter@know-center.at

Abstract. Recently, Knowledge Management has attracted a lot of attention in computer science. However, knowledge management is not just about software tools but also – and maybe even more – about people and organizational cultures. Unlike in software engineering, developers at all three levels (A, B, C) do not only need technical know-how but also organizational and solution know-how to develop successful knowledge management software. This paper argues that personalization concepts closely linked to our notion of knowledge are a promising approach to overcome today's barriers in knowledge management. Instead of focusing on system features, our notion of personalization focuses on the adaptation of metadata, content and structure in knowledge management, thus opening up solution-oriented ways of thinking to A- and B-developers of knowledge management software.

1 Introduction

It is widely recognized that the transitions to an information society and a global knowledge economy will be the most important social and economic changes of the next decade. The global knowledge economy with its high innovation speed and an increasing demand of knowledge intensive products and services calls for new management tools and methods. Therefore, efficient knowledge management has become imperative for almost all types of organizations.

Knowledge management can be addressed from two different perspectives. The first perspective places the emphasis on information technologies as enabling technologies. The second perspective is more people-oriented as it focuses on people and organizations. The difference between these two perspectives is the level at which knowledge management is applied.

The objective of technology-oriented knowledge management is to support knowledge workers at an operational level. That is, information technologies are used to provide the knowledge somebody needs to perform a specific task as well and as efficiently as possible. Often, this requires a careful and smooth integration of knowledge management tools with business process management tools.

In people-oriented knowledge management, the focus is on the people and the organization rather than on the technology. People-oriented knowledge management tries to find answers to questions such as "How can we improve our communication culture?", "How can we manage our human capital more efficiently?",

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 29–41, 2003.

"What methods and incentives exist to foster knowledge sharing and transfer in our organization?" In addition, impact assessment studies are part of people-oriented knowledge management. Typical questions in this context are "How do information technologies change an organization's communication culture?", "How do the employees of an organization get along with the new tools and the possibilities they offer?".

This paper addresses technology-oriented knowledge management from two different perspectives. The industry perspective analyses the role of computer scientists in the knowledge management software industry and how vendors of knowledge management software address the topic of knowledge management. A more scientific perspective identifies weaknesses of current knowledge management software (and their developers). In this context it is argued that possibilities for personalization have a great potential for improving knowledge management software. We also show that the "B" level of Engelbart's notion "A", "B" and "C" level work holds great potential to develop effective and efficient knowledge management software. According to Engelbart, A-level work is directly related to the primary function of an organization. For example, in a software company which designs web interfaces A-level work deals with the development of new web designs. B-level work supports this primary work at the A-level. In the above software company example, B-level work includes the development of WYSIWYG-editors or other tools that directly support the A-level work. The third level, the C-level, provides support for the support at level B. In the above example, this would include development tools and infrastructure frameworks which for example support the development of WYSIWYG-editors.

The paper is structured as follows: The next section briefly introduces our notion of knowledge management and knowledge and stresses weaknesses of today's objectives of technology-oriented knowledge management. Section 3 analyses the role of A- and B-level workers in knowledge management software companies. Section 4 introduces the notion of personalization. Section 5 takes a more scientific view and shows how concepts to personalize content and structure can significantly improve the value of knowledge management software. The paper is concluded in Sect. 6.

2 Knowledge and Knowledge Management

Knowledge and Knowledge Management are two terms for which many different definitions exist depending on the scientific community. In philosophy knowledge is identified, classified and valid information [4]. In information science knowledge is often defined as information in contextualized action [5]. In organization science a separation between explicit and implicit knowledge is commonly accepted [12]. Explicit knowledge includes content which can be formalized and which is based on conscious distinction and selection processes. In contrast, implicit (or tacit) knowledge is based on a person's capabilities to take intuitively decisions and to make intuitively selections and distinctions. Due to its very nature, implicit knowledge can not be codified [9].

For the field of computer science our notion of knowledge is based on a definition provided by Rehäuser [11]:

> Knowledge is the mapping from reality, states and activities onto the internal model of the "real" world, which an individual or an organization has. With this internal model an individual or an organization can make statements about reality.

In other terms knowledge provides a context (i.e., the personal world of an individual; that is how this individual perceives the real world) which helps individuals and organizations to "understand" the real world.

As to knowledge management we do not believe that the term "knowledge management" expresses well what most people mean when they use it. Like love or patriotism, knowledge can not be managed, but one can create environments in which knowledge can develop and flourish. To become more precisely we define knowledge management as follows:

> Knowledge management deals with:
> 1. knowledge-friendly environments in which knowledge can develop and flourish to provide individuals or organizations
> 2. context-sensitive knowledge and
> 3. the ability of knowledge workers to apply the knowledge for action.

The knowledge provision includes the four basic knowledge processes as they are known from literature [7]. This includes developing new knowledge, securing and maintaining new and existing knowledge, distributing knowledge and combining available knowledge. With the provision of knowledge in a given context we ensure that only that knowledge is made available which is necessary to support a knowledge worker to perform a specific task. Finally, knowledge management must ensure that knowledge workers can use and apply the knowledge to perform actions. While the first two items of our definition are closely related to technology-oriented knowledge management, the third item requires methods and concepts from human-oriented knowledge management. The third item also makes the difference to information management which does not deal with building up the capabilities for individuals or organizations to use the information or knowledge provided.

As mentioned above, the provision of knowledge comprises securing and maintaining new and existing knowledge. In this context often the term organizational memory is used. The notion of organizational memory has been around many decades and a common understanding is that not only the explicit and formalized knowledge is part of the organizational memory but also the knowledge in the minds of the individuals working in an organization.

The ultimate goal in today's knowledge management research is to investigate how computer systems can be used to realize organizational memories. The idea is to derive as much organizational knowledge (i.e. the knowledge stored in an organizational memory) as possible from the knowledge in the minds of the individuals. However, one drawback of this approach is that much of the

individual and personal knowledge gets lost as an organizational memory generalizes the individual knowledge. Examples for this generalization include the mandatory use of pre-defined metadata to describe the knowledge or pre-defined repository structures in which each individual's knowledge must be integrated. The result is that the organizational memory only represents one internal and generalized model of the real world. If employees in an organization do not get along with this internal model of the real world the organizational memory is often of very limited value to them. The following figure illustrates this.

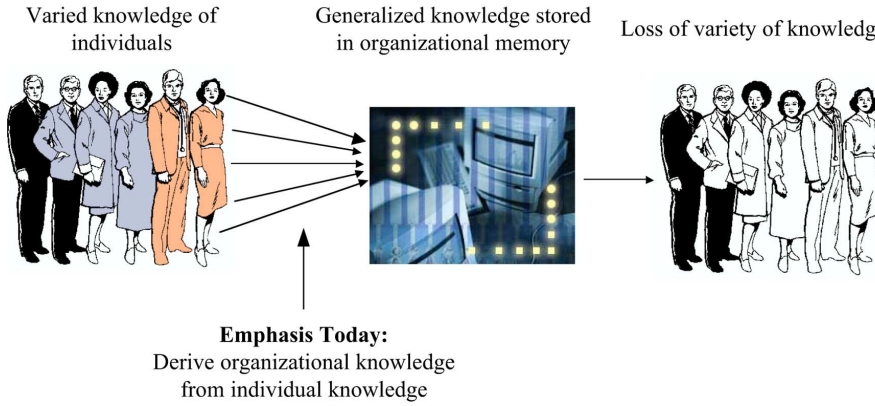


Fig. 1. Illustration of loss of a variety of knowledge due to today's emphasis

For the future the emphasis should also be placed on the second step in the process that is how can individual knowledge be derived best from organizational knowledge. The idea is that organizational memories do not reflect generalized knowledge but different personalized knowledge. This would allow different individuals to use that knowledge which fits best to the internal model they have from the real world. The Fig. 2 depicts this idea.

Section 5 presents how these limitations can be overcome with personalization concepts for knowledge management.

3 A- and B-Level Workers in Knowledge Management Software Companies

In the late 90's industry became aware that the knowledge of their employees is an important asset which significantly contributes to the a company's market value. Rapidly, intellectual assets developed as third value in addition to the traditional organization values capital and labor. At this time also the function of a "Chief Knowledge Officer" (CKO) was introduced in many companies. The CKO was responsible for a company's knowledge strategy, which serves the purpose to increase the intellectual asset and, thus, the market value of a company.

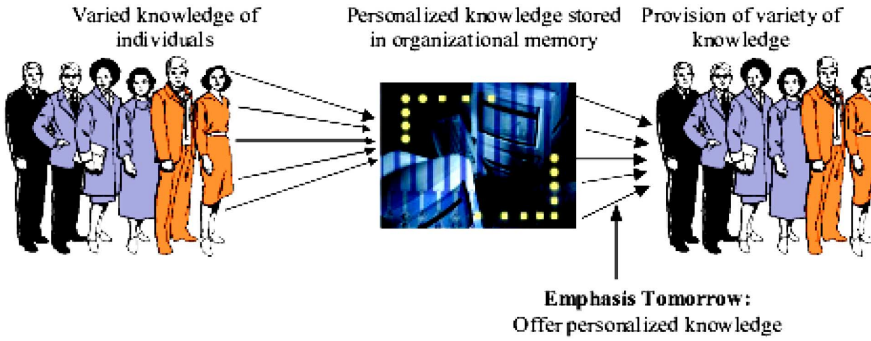


Fig. 2. Offer of personalized knowledge as a challenge for knowledge management

In parallel to these developments vendors of knowledge management software appeared on the market. Thanks to the existence of the CKOs of their customers these companies were in the lucky situation that they could concentrate all their resources at all three work levels (A, B, and C) on technical aspects of their products only. All "soft skills" about "knowledge" and "knowledge management" was provided for free by the CKO. The CKO served as a mediator between the technical oriented people of the knowledge management software vendors and the customers who wanted to introduce and use the software.

This situation coined very much today's knowledge management software. The business model of knowledge management software vendors was (and often still is) to develop generic software which can be adapted according to a customer's need in cooperation with the CKO of this customer.

Today the situation is very different from the situation we had a couple of years ago. Due to the crises in the new economy the CKOs disappeared, they were strategic staff and, thus, often considered as non-productive or overhead. This has a very negative impact on the knowledge management software vendors: In the "good times" they did not build up soft skills in knowledge management, particularly not at the A-level. As a consequence, today technical oriented A-level workers of knowledge management software companies sell the software to potential customers or at least support the sales staff. And they try to sell the software with their language which is very feature-based and not driven by user needs or usage scenarios. The lack of soft skills in knowledge management also explains why vendors sell the software but do not support the introduction process. This is a big problem for organizations which buy such software, simply because they believe that they introduce knowledge management with the purchase of knowledge management software. All too often they must recognize that this is not the case, all what they have is an installation of knowledge management software but no knowledge management.

What is required for the next future is that knowledge management software companies offer better consultancy and more services to their customers. This need has already been recognized by IBM which strengthened their soft skills

with a $3.5 billion take-over of PriceWaterhouseCoopers' consultancy. In addition, a shift from the provision of system features to the provision of knowledge management solutions is required. This shift is more difficult to achieve as those who know the software best, i.e., the staff at A- and B-level, must be involved in this process.

4 Personalization Concepts

Personalization has recently attracted a lot of attention in industry. One reason for this is that several industry studies have revealed the business value of personalization. A Jupiter study of 25 research sites found that web site personalization increased the number of new customers by 47% and revenues by 52% in the year following its introduction [2]. McKinsey's results confirm the value of personalization for an eCompany [2]. For example, thanks to higher transparency in order status, pricing and order fulfillment costs for customer servicing could be reduced by up to 58%, the repeat-purchase rates increased by 25% to 45%, and explicit and implicit customer communications could significantly improved.

In industry the following definitions for personalization are common. The Personalization Consortium [8] defines personalization as follows:

> Personalization is the use of technology and customer information to tailor electronic commerce interactions between a business and each individual customer.

This definition relates personalization to the system features and the content provided by the respective systems.

Bea Systems [1] is a vendor of a server software supporting personalization. Bea defines personalization as follows:

> BEA WebLogic Personalization Server helps businesses enrich the visitor experience using rules-based implicit and explicit personalization. Implicit personalization adapts the site by measuring visitor interactions and improving the online experience while visitor identity is unknown. Explicit personalization further adapts the site for known visitors using online profile and offline visitor databases.

Similar to the definition of the Personalization Consortium the main focus is laid on the functional system components (adapts the site by ... measuring visitor interactions and improving the online experience) and the content (personalization ... adapts the site).

Also in scientific definitions of the term personalization the emphasis is placed on system features and content. For example, Mobasher et al. define personalization as follows [6]:

> Web personalization can be defined as any action that tailors the Web experience to a particular user or set of users. The experience can be

> something as casual as a Web site or as (economically) significant as trading stocks .... The actions can range from simply making the presentation more pleasing to anticipating the needs of a user and providing customized information.

Taking the above definitions of personalization into account, we propose the following definition which puts the term personalization context with the above introduced notion of the term "knowledge" in computer science:

> Personalization of a system is the adaptation of:
> – its system features,
> – the content managed by the system and
> – its structural components for organizing content
> according to the internal model of reality, states and activities system users have.

In other terms this definition says that the better the personalization is the better is the fit between a user's mental model of the real world and the real world. Or even more simplified: personalization helps users to "understand" reality.

5 Personalization in Knowledge Management

As outlined in Sect. 3, the staff – particularly at the A-level – of today's knowledge management system vendors still think too feature-based. This bears the risk that the application of personalization concepts in knowledge management focuses on the feature level only. Instead, we suggest to place the emphasis for personalization concepts in knowledge management on the metadata, the content and the structure. This focus is beneficial for two reasons: Firstly, knowledge management is more about content and structure rather than on features. Secondly, it is easier to avoid that A-level workers think feature-based if the underlying basis – which is the conceptual design carried out by B-level workers – is not feature-based.

The subsequent sections briefly describe the personalization concepts for metadata, content and structure.

5.1 Personalization of Metadata

Metadata is commonly understood as data about data. A metadata item consists of a metadata name (e.g., "topic") and a metadata value (e.g., "computer science"). Metadata is a set of such name value pairs.

With this notion of metadata, three different possibilities exist to personalize metadata: (1) the metadata name can be personalized, (2) the metadata value can be personalized, (3) both, the metadata name and the metadata value can be personalized. The following figure (3) shows an examples for each possibility.

For the sake of completeness it should be mentioned that also multiple personalizations for one metadata item are possible [13].

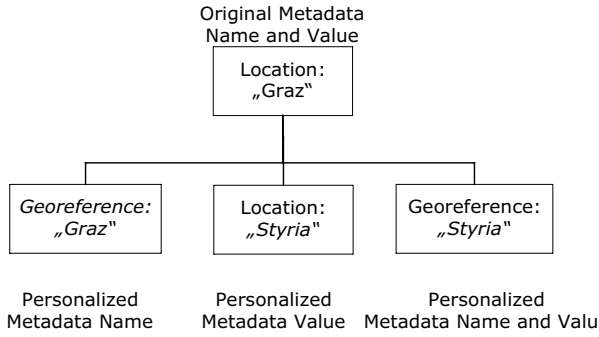


Fig. 3. Options for personalization of metadata

5.2 Personalization of Structure

The personalization of the structure of a knowledge space allows users to organize the content along many different dimension (e.g., an organizational dimension, a thematic dimension etc.).

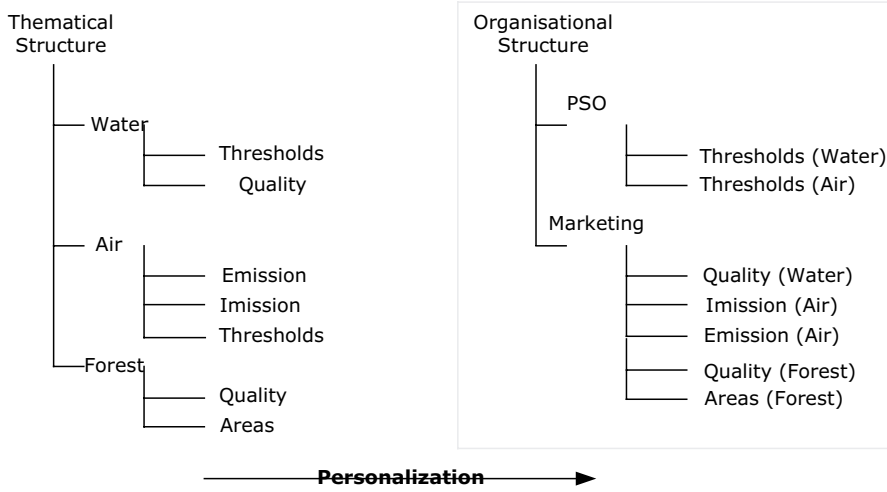


Fig. 4. Example for personalized structures

The above figure depicts this idea: While the original structure on the left is organized along thematic fields, the personalized structure on the right reflects the different departments of an organization.

5.3 Personalization of Content

We differentiate between the personalization of content and personal content. The personalization of content is always related to the content which is available for all users of a knowledge management system. In contrast, personal content is content which is only accessible to an individual and to which other users do not have access to.

The personalization of content allows to hide parts of the content or to change its layout according to different user's needs. The following figure shows an example for the first case. While the original content consists of some text and a graphic, the personalized content contains the text only.

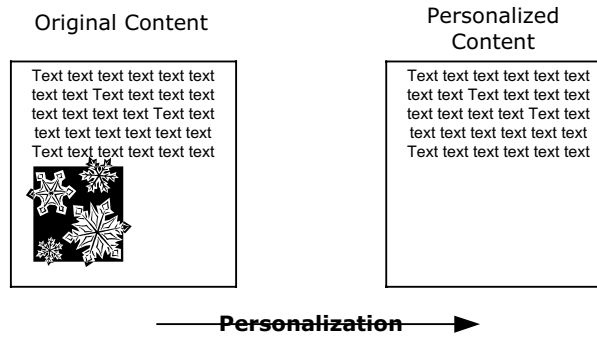


Fig. 5. Example for personalized content

With personal content users are offered with the possibility to extend a knowledge space with content they need to perform their tasks. The personal content is inserted in the structure of the knowledge space according to the user's needs. Figure 6 shows an example of personal content in knowledge spaces.

For more details the interested reader is referred to [13].

5.4 Modeling Personalization and Generic Software Architecture

This section briefly presents how the above concepts can be modeled in UML.

The following figure shows the UML diagram for personalized metadata.

Figure 7 shows that a knowledge space can contain an arbitrary number of knowledge objects (a knowledge objects represents content and can be a text, a graphic, a video, an audio etc.). For each knowledge object one set of original metadata exists. In addition, the metadata for a knowledge object can be personalized, the optional character of the existence of personalized metadata is expressed by the 1-to-0..1 relationship. Personalized metadata is stored separately form the original metadata. The reason is that often original metadata catalogs can not be modified by arbitrary users.

To model personalization of structure, a clear understanding of the relationship between knowledge objects, knowledge spaces and structure is required.

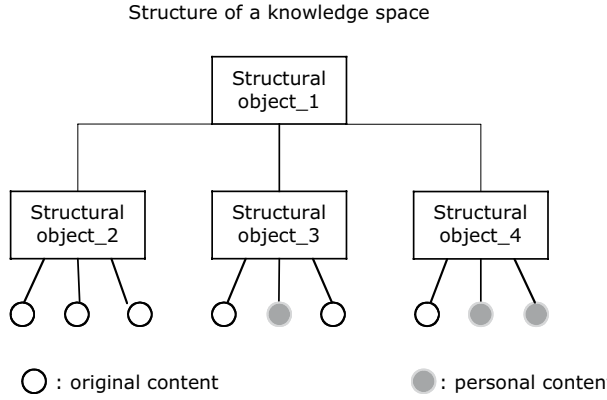


Fig. 6. Example for personal content

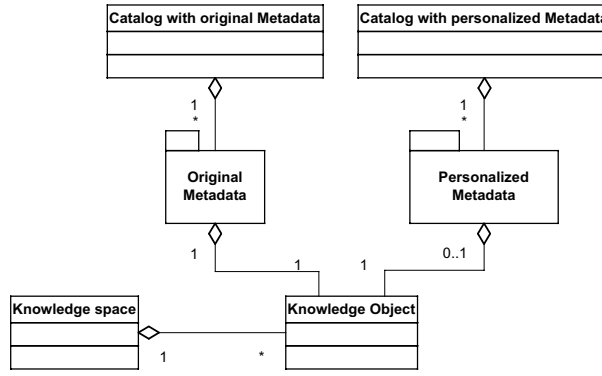


Fig. 7. UML-diagram for personalized metadata

In our notion, a knowledge space is made up of many different structures. The actual structure (e.g., hierarchy, directed graph etc.) is defined by structural objects and the relationships between them (e.g., is-contained). Each knowledge objects can belong to arbitrary many structures. The following UML diagram shows this idea. It is worth mentioning that the UML diagram does not define the type of the structure.

With this in background, personalization of structure can be modeled in UML as follows: A knowledge object can either belong to a (original) structural object or to a personalized structural object. A personalized structural object can be part of a personalized structure (e.g. if a user defines his own hierarchy of folders) or it can be part of the original structure (e.g., if a user has just added a personal folder in the existing folder structure).

Finally, we address personal content and personalized content by adding the concepts of personal knowledge object and personalized knowledge object to the above diagram. As Fig. 10 shows, a personal knowledge object can only exist

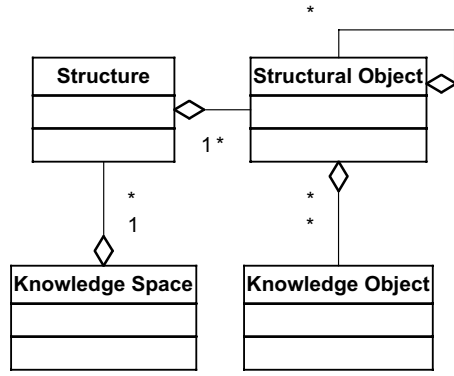


Fig. 8. UML-diagram for structures in knowledge spaces

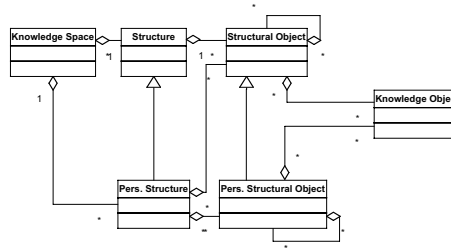


Fig. 9. UML-diagram for personalized structures in knowledge spaces

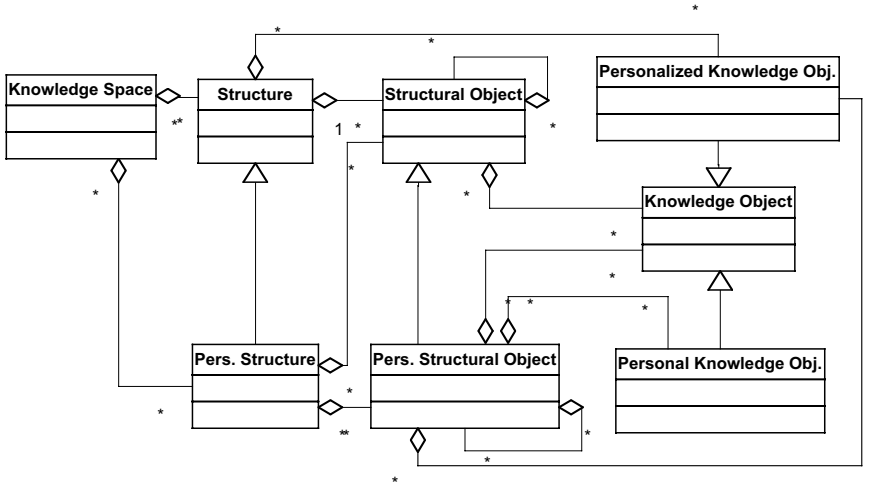


Fig. 10. UML-diagram for personalized knowledge objects and personal knowledge objects

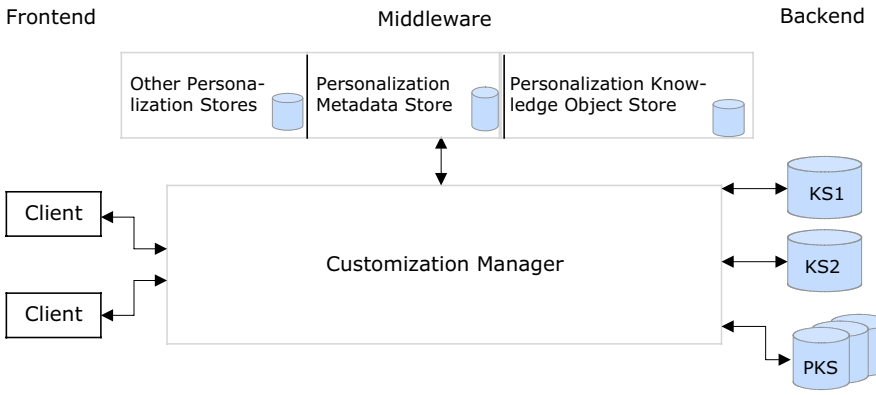


Fig. 11. Generic Software Architecture for Personalization

in a personal structural object and not in original structural object. With this restriction we avoid that users extend the original knowledge space with personal content which is not relevant to others. A personalized knowledge object inherits its behavior from "its" original knowledge object. As a personalized knowledge object always exists in the context of an original knowledge object it can exist in both, the original structure and the personal structure.

With these diagrams in mind we now want to discuss a possible software architecture for systems which support personalization. The development of the PADDLE system, a personal adaptable digital library environment [3], revealed that personalization concepts are best provided in a middleware component of a system. The front-end of such a system covers all user-interface aspects while the back-end manages all original systems.

As depicted in Fig. 11, the Customization Manager is positioned in the middleware component between the clients and back-end systems. The Customization Manager offers all algorithms which are needed for personalization. For example, it offers the users the algorithms to define personalized metadata and to search for knowledge objects in the back-end knowledge spaces KS1 and KS2 using personalized metadata.

Based on this generic architecture we developed a specific architecture serving all needs and requirements in the context of the PADDLE system. Also, the PADDLE system has implemented all personalization concepts introduced in this paper. Several applications of PADDLE in industry environments proof the usefulness of personalization concepts. For more information, use cases and further application scenarios, the interested reader is referred to [13] and [3].

6 Conclusion

In the future only those knowledge management software vendors will remain successful who can offer knowledge management solutions (and not only systems) to their customers. This, however, requires a new mind set for A- and

B-developers who still think too much in terms of system features. With this in background, this paper introduced a definition of personalization which is closely related to our notion of knowledge. This link opens up new perspectives of thinking: instead of focusing on mere technical features, the emphasis can now be placed on personalization concepts for metadata, content and structure – the most important ingredients of knowledge management.

Acknowledgements

The Know-Center is a Competence Center funded within the Austrian Competence Center program K plus under the auspices of the Austrian Ministry of Transport, Innovation and Technology (www.kplus.at).

References

1. Bea Systems. 2002. `www.bea.com`.
2. Broadvision. 2001. Business Value of Personalization. `www.broadvision.com`.
3. Hicks, D., Tochtermann, K. 2001. Personal digital libraries and knowledge management. *Journal of Universal Computer Science 7*(7), 550-565.
4. Hubig, C. 2001. Philosophy. In [10] vol. 3, 57-67.
5. Kuhlen R. 2001. Implicit knowledge from the perspective of information science (in German). In [10], vol. 3, 69-81.
6. Mobasher, B., Cooley, R., Srivastava, J. 2002. Automatic personalization based on web usage mining. *Communications of the ACM 43*(8), 142-151.
7. Nonaka, I., Takeuchi, H. 1995. *The Knowledge-Creating Company*, Oxford University Press, New York.
8. Personalization Consortium. 2002. `www.personalization.org`.
9. Polanyi, M. 1985. *Implicit Knowledge*, Suhrkamp-Verlag, Frankfurt.
10. Radermacher, F. J., Kämpke, T., Rose, T., Tochtermann, K., Richter, T. 2001. Management of implicit knowledge: Learning from nature (in German). Study for the German Ministry for Education and Research, FAW Ulm, (`www.faw.uni-ulm.de`).
11. Rehäuser, J., Krcmar, H. 1996. Knowledge management in organizations (in German). *Managementforschung 6: Wissensmanagement*, Schreyögg, G., Conrad, P. (Eds.), de Gruyter Verlag, Berlin.
12. Schwaninger, M. 2001. Implicit knowledge and management (in German). In [10], vol. 3, 149-164.
13. Tochtermann, K. 2002. Personalization in the context of digital libraries and knowledge management. Post-Doctoral Thesis, Graz University of Technology.

Babylon Bookmarks: A Taxonomic Approach to the Management of WWW Bookmarks

Nikos Karousos[1,3], Ioannis Panaretou[1,2], Ippokratis Pandis[1,3], and Manolis Tzagarakis[3]

[1] Computer Engineering and Informatics Department
Patras University, Rio, Patras, Greece
pandis@ceid.upatras.gr
[2] Business Management & Administration Department
Patras University, Rio, Patras, Greece
panareto@ceid.upatras.gr
[3] Computer Technology Institute
Patras, Greece
{karousos, tzagara}@cti.gr

Abstract. Taxonomic reasoning can be a useful organization paradigm to address issues dealing with the management of WWW bookmarks. Management of WWW bookmarks comprise one of the most frequent activities of WWW users. In this paper, "Babylon Bookmarks" are presented allowing users to apply taxonomic reasoning on WWW bookmarks. "Babylon Bookmarks" is based on the "Babylon System," an infrastructure to support general taxonomic reasoning tasks. The "Babylon System" allows the development of arbitrary taxonomic services based on Open Hypermedia Specifications.

1 Introduction

Taxonomic reasoning is a specific kind of reasoning which deals with categorization and comparison of similar piece of information. In such cases, the analyst is trying to "see" the information in terms of grouping them in a set of similar notions (by a way of an "is-a" relationship), rather than associating them exactly [7]. Issues associated with this way of managing information, define the hypertext categorization area as well as the hypertext categorization systems (systems incorporating taxonomic reasoning). Taxonomic reasoning leads to a hierarchical, tree-structure organization for sets of information. We often encounter hypertext systems supporting taxonomic reasoning in botanical work, linguistics' and other relevant sciences' activities [4,7,6]. The taxonomic reasoning approach can be beneficial to various problem domains.

On the other hand, Bookmark Management is one of the most important problems that WWW users have to deal with. The implementation of several Bookmark management systems did not clearly provided efficiency since users heterogeneous requirements could not be met in a single system. Advanced categorization methods need to be applied in order to provide satisfactory capa-

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 42–48, 2003.

bilities. Hypermedia systems use taxonomies aiming to give answers to relevant information organization tasks [4].

This paper presents the design of "Babylon System" which provides a suitable set of tools for the management of taxonomies, based in Open Hypermedia System (OHS) [3] principles. Next, "Babylon Bookmarks," a real-world service for WWW Bookmark management is created, using the Babylon System. Babylon Bookmarks was designed to provide improved solutions for some of the most important bookmark management problems noted in literature like the resource discovery, the bookmark recall and the bookmark maintenance problem [2].

The idea is that the bookmark management and sharing problem could be reduced to a taxonomy management problem. In this way, Open Hypermedia features can be applied to the bookmark management problem providing several advantages both for developers and users. According to this idea Babylon grants developers with the following capabilities: a) to utilize taxonomic reasoning advantages approaching with a better way relevant problems, b) to develop a shared bookmark management system easy and fast and, c) to be able to construct other specific categorization services.

2 Approaching Bookmark Management as a Specific Taxonomic Reasoning Issue on OHSs

Bookmarks are used as "Personal Web Information Spaces" helping people remember and retrieve interesting web pages. Despite the fact that several bookmark management systems have been proposed, none of them can fully correspond to the requirements so as to provide efficiency to the users. Particularly, the main bookmark management problems as defined in [2] are:

1. The discovery problem: the problem of the automatic discovery of new interesting web sites for specific-interest group of users.
2. The bookmark recall problem: the fast access to stored bookmarks.
3. The bookmark maintenance problem: the bookmark revision that matches the evolution of the user information needs as well as to ensure that stored bookmarks are still valid.

Many alternative methods for bookmark organization have been presented lately, but two are the main ways that users think of bookmarks: as sets of information and as hierarchies [1]. The hierarchical bookmark data model could be applied in Open Hypermedia System supporting taxonomies and the reason is that a bookmark could easily be considered as an item with specific characteristics (e.g. URL, title, comments etc.) belonging to a single taxonomy.

Items of any data type (document, video, sound), described by a number of characteristics, are fundamental concepts in terms of categorization. Each item can be placed only in a certain category. Any item can be inserted or removed from a category. Categories are not referenced by their content but they can be described by an unlimited number of characteristics. The set of items and

categories define a categorization (or taxonomy) representing a hierarchy (or tree-structure).

Thus, constructing Babylon as a system that supports taxonomies and applying it to "build" a specific bookmark management service (Babylon Bookmarks) enriches the system with OHSs taxonomic advantages. Additionally, Babylon allows developers to reuse the framework in order to create other categorization services easy and fast.

3 The Babylon Project

The Babylon project (the name was inspired by the hanging gardens of Babylon) is based on the idea to create an integrated framework aiming to provide multiple categorization services using abstractions met in taxonomies. The main goal is to provide developers with an easy and fast way to construct Internet services for item categorization.

The model has the following characteristics: The main entity of the data model is the object (item) in which several characteristics are assigned. Each item can be inserted in a single category. Each category is identified by a group of characteristics and may contain a set of items or other categories, formulating a tree-structure. The system supports the creation of category shortcuts in other categories of a tree, giving the notion of relevance. Finally, the meaning of association is also defined; a specific relationship between different tree categories that are being used for automatic tree-to-tree item transferring and sharing.

During the design and implementation of Babylon, we focused on two innovating points: a) user management and category sharing capabilities and, b) association capabilities between similar categories of different taxonomies, targeting on automatic update of all the common interest categories. This feature allows end users to get notified when associated categories of other taxonomies change (e.g. when items are added or deleted).

The targeted task is to redefine Babylon System should as an integrated component supporting taxonomic services in Component Based Open Hypermedia Systems (CB-OHS) [3] such as Callimachus [8].

3.1 Architecture

Babylon's architecture consists of four layers: the storage layer that is responsible for storing and managing the structural and non-structural information imported into the system, the taxonomic management layer that provides taxonomic creation and manipulation services, the Internet Service layer consisting of several network tree-structure administration tools for the organization and management of specific information and, the user layer that supports the final services for end-users.

The storage layer and the taxonomic management layer comprise the basic infrastructure that provides specific taxonomic services for developers.

3.2 Storage Layer

The storage layer is the system's repository, and it is composed by the structural storage system and the data storage system. The structural storage system stores and organizes the structural information while the data storage system is responsible for the user-imported data. The system's storage layer can incorporate any relational DBMS to provide stronger query capabilities, performance and reusability. Both the data and the structure storage systems can either coexist at the same RDBMS or be distributed at different RDBMS's.

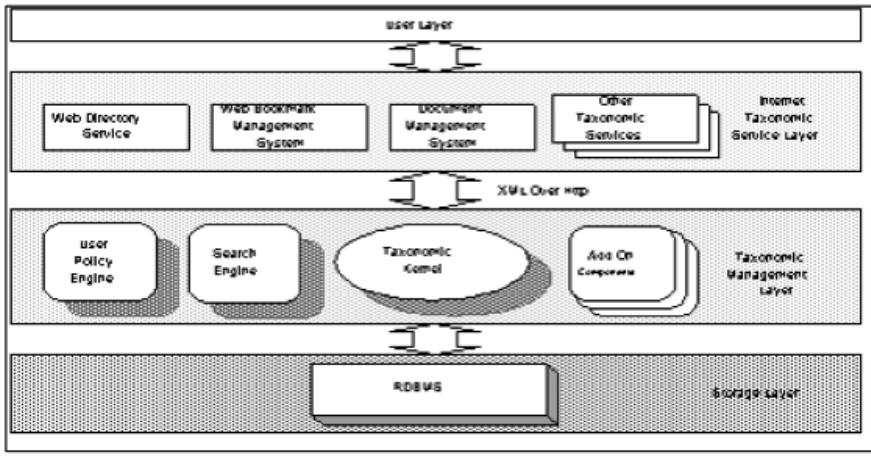


Fig. 1. Babylon's architecture. The level division and the services naming was made according to the CB-OHS architecture in order to emphasize the easily transformation of the system to be able to work as a CB-OHS service

3.3 Taxonomic Layer

The taxonomic layer provides basic taxonomic management capabilities to the system, by serving structure and content management as well as sharing and comparison requests. The layer kernel is a taxonomic structural server, which is responsible for the taxonomic structure creation, management and storing as well as the data categorization requests. The structural server is also cooperating with a set of specific-task components like search engines, user-policy engines, etc. The communication with the Internet Service layer, which is the next layer in our proposed architecture, is achieved using the HTTP protocol, sending the data in a very simple XML form.

The taxonomic layer's infrastructure is flexible and open. Its openness is due to the ability to cooperate easily with any external specific-task component. Its flexibility stems from the fact that the kernel is constructed in such a way

to allow modifications and additional operations. This layer is also enriched with additional tools, helping developers to create services quickly and easily. For instance, assistants (wizards) for the creation of an integrated taxonomic-management interface as well as the automatic web-page creation of a taxonomy-navigation tool (e.g. generating automatically PHP code) have been enhanced.

Finally, our main future goal is to develop an integrated service based on CB-OHS architecture. The system should be able to add and manipulate other hypertext scenarios (e.g. navigational, spatial etc.) over taxonomies [5,9].

3.4 Internet-Service Layer

This layer is constructed by a set of tree-management visual tools. These tools are offered either directly to the users as client applications or as web-based services. Also, a number of association-mapping tools between different categories as well as sharing tools for the administration of user permission policies (e.g. which categories to be shared with particular users, etc.) are introduced. Thus, the Internet Service Layer can interact with users as specific-task web sites (like Web directory service providers and on-line bookmark management services) or as services for intranet document-management services, and as other services derived from users' needs.

3.5 User Layer

The User Layer enables the users to benefit from the services provided, either by using their own web browser or a stand-alone client application. These applications include visual representations of the taxonomic structures and a set of tools that provide capabilities for the management, searching and viewing of the data sets.

4 Babylon Bookmarks: Using Babylon for Real-World Services – Creating an On-line Shared Bookmark Management System

The creation of a bookmark management system based on Babylon's infrastructure is an easy task. It is required the first two layers to be installed in a network supporting TCP/IP. First, the developer must define a bookmark pattern that is a set of characteristics describing a bookmark (e.g. URL, Title, Comments etc.) In addition, the developer must create the user interface (Internet Service Layer), giving to users the ability to manipulate their data within hierarchies and to share their bookmarks and associate categories. This could also be done using Babylon's specific-task assistants (wizards). Furthermore, the developer can create add-on tools for the users as well as user account management tools for administrators.

With respect to visualization and user interaction issues in Bookmark Management Systems, Babylon Bookmarks architecture provides effective solutions to the issues presented in Tables 1 and 2.

Table 1. Open problems [2] and Babylon's approach

Bookmark System Problems that need to be handled	Babylon Bookmarks Approach
The resource discovery problem	Association capabilities among categories that provide direct notification when a new URL is inserted in other users associated common interest categories.
The bookmark recall problem	Providing the user either with web-based tools or with stand-alone client applications that help him create personal tree structures and shortcuts on categories, seek and search time are drastically reduced.
The bookmarks maintenance problem	The adjustment of a dead-link trace engine at Babylon's taxonomic layer contributes to the elimination of the problem.

Table 2. Shared Bookmark System Drawbacks and Babylon's approach

Shared Bookmark System Drawbacks	Babylon Bookmarks Approach
The weak customization of shared spaces The lack of using privacy protection policies	Sharing and privacy issues comprise an integral part of Babylon's functionality, providing category sharing and association capabilities as well as user groups with certain permissions per group and user.
The bad adoption of centralized architectures	Despite the fact that Babylon's architecture describes a single structural server's existence, it is possible to enhance full distribution at the current layer, providing a fully distributed bookmark management system.

5 Conclusions

In this paper, we presented "Babylon," a framework that provides a set of tools for management of taxonomies and "Babylon Bookmarks," a real-world service for WWW Bookmark management. The proposed Babylon architecture offers important advantages like: a) system distribution, openness and scalability, b) easy creation and management of categorization services like bookmark management services by developers c) advanced sharing and associating features and d) automatic resource discovery (using associated categories).

Based on the Component-Based Open Hypermedia Systems' requirements, the taxonomic approach for categorization services provides additional capabilities both for developers and users and gives answers to many open problems in a wide range of information organization areas.

References

1. Abrams, D., Baecker R., and Chignell, M. 1998. Information archiving with bookmarks: Personal Web space construction and organization. *Proceedings of ACM Conference on Human Computer Interactions (CHI'98)*, (Los Angeles, CA, Apr), 41-48.
2. Kanawati, R., Malek, M. 2000. Informing the design of shared bookmark systems. *Proceedings of RIAO2000, Paris, France*, 170-180.
3. Nürnberg, P. J., Leggett, J. J. and Wiil, U. K. 1998. An agenda for open hypermedia research. *Proceedings of the '98 ACM Conference on Hypertext* (Pittsburgh, PA, Jun), 198-206.
4. Nürnberg, P. J. 1997. *HOSS: An Environment to Support Structural Computing.* Ph.D. dissertation, Department of Computer Science, Texas A&M University, College Station, TX.
5. Nürnberg, P. J. and Leggett, J. J. 1997. A vision for open hypermedia systems. *Journal of Digital Information 1*(2) (Special Issue on Open Hypermedia Systems).
6. Parunak, H. V. D. 1993. Hypercubes grow on trees (and other observations from the land of hypersets). *Proceedings of the 5th ACM Conference on Hypertext (Hypertext '93)* (Seattle, WA, Nov), 73-81.
7. Parunak, H. V. D. 1991. Don't link me in: Set based hypermedia for taxonomic reasoning. *Proceedings of the 3rd ACM Conference on Hypertext (Hypertext '91)* (San Antonio, TX, Dec), 233-242.
8. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., Schraefel, M., Vaitis, M., Christodoulakis, D. 1999. Structuring primitives in the Callimachus component-based open hypermedia system. *Proceedings of the First Workshop on Structural Computing* Technical report CS-99-04, Aalborg University Esbjerg, Computer Science Department.
9. Østerbye, K. 1999. Hierarchical structure through navigation side bars. *Second Workshop in Hypermedia Development: Design Patterns in Hypermedia.*

Approaching Wordnets through a Structural Point of View

Dimitris Avramidis, Maria Kyriakopoulou, Manolis Tzagarakis, Sofia Stamou, and Dimitris Christodoulakis

Computer Engineering and Informatics Department
University of Patras, GR-265 00, Rion Patras, Greece
Research Academic Computer Technology Institute
Riga Feraiou 61, GR-262 21, Patras, Greece
{avramidi, kyriakop, tzagara, stamou, dxri}@cti.gr

Abstract. The analogy of a semantic network to hypertext has long been recognized, and semantic networks have been considered as a logical model of hypertext – especially for those hypertexts with typed nodes and links. Moreover, wordnets form the most representative type of semantic networks in the field of Natural Language Processing and semantics in particular. It is obvious that hypertext and wordnets share many common points regarding their fundamental principles and the objectives towards which they both aim. This paper expresses our initial thoughts towards incorporating the Balkan WordNet in Callimachus CB-OHS, as such systems can conveniently support structure. We strongly believe that such task can be fulfilled by using already implemented domain abstractions along with a new set of behaviors.

1 Introduction

Hypertext has always been closely related to the idea of freedom to associate, making it to be considered as an alternative means of structuring information. This new promising field provides its users (namely, authors and readers) with effective ways of presenting and exploring information. For authors, hypertext systems offer a high degree of flexibility for connecting pieces of information and presenting it as an assembled collection in an information network. For readers, hypertext provides tools for navigating in these information networks and for exploring them freely. Therefore, hypertext can be a precious dialogic means, facilitating the organization of information according to the user needs.

On the other hand semantic networks form a highly structured linguistic resource enabling a flexible navigation through the lexical items of a language. Wordnet forms a kind of conventional dictionary where semantic information of the terms it contains is represented. The main structural entities of wordnets are language internal relations through which words are linked based on their semantic properties. The main contribution of wordnets in lexicography is the systematic patterns and relations that exist among the meanings that words

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 49–57, 2003.

can be used to express. In this respect wordnets as a particular type of semantic networks resemble much hypertext as far as the structural organization of information is concerned.

This paper is a first attempt to model wordnets using Callimachus CB-OHS [25]. Moreover, the application of CB-OHSs in the linguistic domain, would be a very good and tough exercise for these systems. The degree to which CB-OHSs can solve problems, in such a structure oriented domain as linguistics, in a convenient and efficient way can give us important insights about CB-OHSs in particular, and structural computing in general. The challenge is even greater if one considers that structural computing has been inoculated with philosophical ideas from structuralism, a descendant of linguistics.

Our motivation was BalkaNet[1] (**Balka**n Word**Net**), an IST project undertaken by our laboratory, that aims at combining effectively Balkan lexicography and modern computation. The most ambitious feature of the BalkaNet is its attempt to represent semantic relations and organize lexical information from Balkan languages in terms of word meanings. One envisaged application of the BalkaNet concerns its incorporation in Information Retrieval (IR) systems in order to support *conceptual* text retrieval as opposed to exact keyword matching. Trying to realize conceptual indexing, computations over wordnet structure must be applied, a task that can be better facilitated by CB-OHSs.

2 Structure in Semantic Networks

Wordnets form the most representative type of semantic networks in the field of Natural Language Processing and semantics in particular. Motivated by theories of human knowledge organization, wordnet emerged as a highly structured language repository, where words are defined relatively to each other. Unlike machine-readable dictionaries and lexica in book format, wordnet makes the commonly accepted distinction between conceptual-semantic relations, which link concepts and lexical relations, which link words [5]. Thus, despite their resemblance to typical thesauri, wordnets in general clearly separate the conceptual and the lexical levels of language, and such a distinction is reflected via semantic-conceptual and lexical relations that hold among synsets and words respectively. Wordnets form semantic dictionaries that are designed as networks, partly because representing words and concepts as an interrelated system seems to be consistent with evidence for the way speakers organize their mental lexicons [18,10].

Wordnets' hierarchical structure–as shown in Fig. 1–allows a searcher to access information stored in lexical chains along more than one path, semantics being among them. Conceptual structures are modeled as a hierarchical network enabling a graphical representation of the lexicalized concepts when the latter are denominated by words [24]. The theoretical analysis shows dependencies among semantic relations, such as inheritance of relations from sub-concepts to

[1] For more information please visit `http://www.ceid.upatras.gr/Balkanet`.

super-concepts. Therefore, related senses grouped together under the same lexical chain form preliminary conceptual clusters. Words belonging to the same lexical chain are connected via language internal relations, each one denoting the type of relation that holds among the underlying word meanings. Some of the language relations are bi-directional in the sense that if a link holds between term A and B then a link also holds between term B and term A. However, bi-directionality of the relations strongly depends on the language particularities and semantic properties of the underlying word meanings.

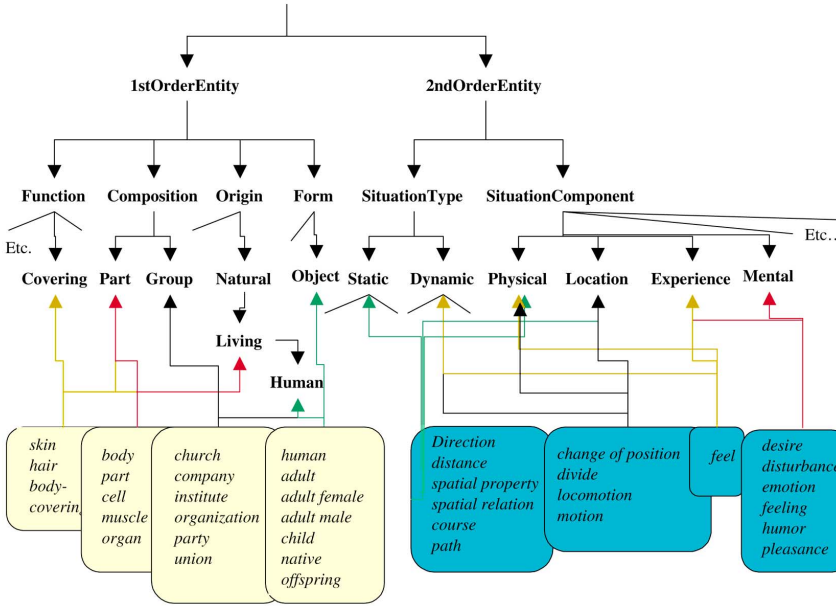


Fig. 1. This is an example of a WordNet depicting only the basic relations of *hyponymy* and *hyperonymy*, each one forming a tree. However, there are many other typed relations, such as antonymy, meronymy, etc., that have been recognized [6]

In order to account for particularities in lexicalized concepts, tags are assigned to each lexical relation denoting specialized semantic characteristics of a word's meaning. Tags can be viewed as a means of semantic constraints posed upon semantic relations that link word meanings rather than word forms. Moreover, tags provide information about which of the semantic properties represented in a lexical chain are inherited to its components. In this respect, words represent an atomic and unbiased level of individuality that becomes meaningful via anchoring of semantic relations. As Hasan [8] pointed out, any word in a chain can be related to multiple other words in that chain. All lexical relations form a graph where cycles are disallowed since after all they contribute very little of any new information.

Summarizing, the structure of lexical data within wordnets is what differentiates the latter from traditional lexicographic aids (both dictionaries and thesauri). The motivation behind construction semantic networks in the form of a graph relies on the fact that lexical data becomes meaningful only via predefined linguistics structures. Navigation through the content of wordnets becomes feasible via language internal relations, which form the main notion around which structure is defined.

3 Approaching Wordnet via Hypertext

Adopting the "primacy of structure over data" [21], hypertext can be seen as a technology well suited to exploring different kinds of representational structures [16]. Viewing different parts of information as objects, users, often referred to as readers, can navigate through it in a more effective and convenient fashion. Additionally, authors can manipulate information according to their needs [11]. Therefore, hypertext can be regarded as an informal mechanism, which describes the attributes of these objects and captures relationships that possibly exist between them. Such a characteristic made hypertext become known as an alternative way of structuring information.

The analogy of a semantic network to hypertext has long been recognized [4], and a semantic network has been considered as a logical model of hypertext – especially for those hypertexts with typed nodes and links. As it is widely known, a semantic network is a knowledge representation scheme consisting of a directed graph in which conceptual units are represented as nodes, and relations between the units are represented as links. The graph becomes semantic when each node and link is assigned a particular type, making it meaningful. The essential idea of semantic networks is that the graph-theoretic structure of relations can be used for inference as well as understanding [14]. In this paper we claim that wordnets, the most representative type of semantic networks, may be supported by a CB-OHS.

3.1 Hypertext and Wordnet: Similarities

Hypertext and wordnets share many common points regarding their fundamental principles and the objectives towards which they both aim. In particular, they are both targeted towards capturing relations that possibly exist between objects and thus providing information of the underlying objects via various types of links used for describing the relations. Therefore, the main characteristic of wordnets and hypertext systems is the ability to create associations between semantically related information items. On the one hand, these associations imply purposeful and important relationships between associated materials, whereas on the other hand the emphasis upon creating associations stimulates and encourages habits of relational thinking of the user [12].

Relations form the notion around which both semantic networks and hypertext are organized. In the case of semantic networks, relations are denoted

explicitly between the lexical units they contain via predefined lexical links, and capture information on the semantic properties of words. In the case of hypertext, although the notion of association can be met in all hypertext domains, the navigational domain with the use of *links* is more closely related to it. Consequently, lexical relations form the fundamental entity of semantic networks the same way as associations in hypertext form the basic structural element around which domains are modeled.

In both cases, information objects (either lexical or not) are heavily structured in order to enable users of wordnets or hypertext navigate through the information they contain successfully. Structure is achieved via internal links, which form the basis on which information is stored and expressed. However, links in semantic networks and hypertext are until recently viewed as two distinct elements and no attempt has been made towards comparing the two. We report on the similarities that exist between hypertext relations and semantic links in an attempt to model the latter in hypertext systems.

In order to support this linking activity in an effective way, hypertext researchers have created a flexible link structure incorporating different levels of functionality. More specifically, in hypertext one can create single or bi-directional links, binary or n-ary links, links to links, automatically activated links, etc. Similarly, links in wordnet are bi-directional and there is generally no restriction on the number and types of links they could be included in it as long as the relatedness between the information items is properly and adequately expressed. Bi-directionality of links indicates that if an object A is somehow related to an object B then object B is again related via the same or another relation to the object A.

However, since bi-directionality might not always be the case in wordnets, special tags need to be attached to the relations to denote their single direction. Namely, tags are being used on semantic network relations to indicate that a lexical item is related to another via a particular type of link but not vice versa. Tags are attached to each link separately and act like constraints on the information provided by the link. However, in the case of hypertext, due to the existence of many specialized domains, the notion of tags is used implicitly.

Furthermore, besides creating associations among semantically related information items, another characteristic shared between hypertext and semantic networks is inheritance. This feature implies that properties of the father are inherited to the children. More specifically, the notion of generalization and specialization forms the principle on which relations are expressed. Specialization and generalization define a containment relationship between a higher-level entity set and one or more lower-level entity sets. Specialization is the result of taking a subset of a higher-level entity set to form a lower-level entity set, whereas generalization is the result of taking the union of two or more disjoint (lower-level) entity sets to produce a higher-level entity set.

Inheritance in wordnets is described via the *H/H tree* (Fig. 1) that is the complementary hypernymy/hyponymy relations. This type of relationship between objects result in viewing wordnets like tree-structured sources of information,

and thus not allowing circular loops. As far as hypertext is concerned, these organizational structures exist in the taxonomic domain under the respective terminology of *supertaxon* and *subtaxon*. The subtaxon is associated with the supertaxon via an "is-a" relationship, inheriting all the characteristics that the latter might have. In particular, the user can classify objects (known as specimens) into sets according to their features, search within the members of a set to find relationships or discreet subsets, and create new sets from the already existing ones.

3.2 Using Callimachus to Model Wordnet

The Callimachus CB-OHS attempts to provide the framework in which different hypertext domains co-exist and structure servers – providing new abstractions and services – for new domains can be developed. Special attention has been given in the provision of suitable tools, which are part of the methodology, to facilitate structure servers' development. One such tool are the structure templates that aim at maintaining the specifications of the structure model of a particular hypertext domain. Structure servers operate guided by these structure templates to provide domain specific abstractions and constraints. In particular, they operate on structural objects transforming them to useful abstractions and allowing clients to use them according to the domain specific restrictions.

Trying to model the Balkan wordnet using Callimachus, its structure must be explicitly defined. Based on the common points that hypertext and wordnets share, we realized that the latter ones borrow many characteristics from different hypertext domains, thus raising issues that have been met in cross domain interoperability topics [1,17]. Taking this observation into account, strong evidence exists in building the BalkaNet structure server on top of already existing structure servers (e.g. navigational, taxonomic) requesting the appropriate services (see Fig. 2).

A specification of an adequate set of behaviors would result in better exploitation of the wordnet structure. In this way, applications built on top of Callimachus, such as conceptual indexing, can perform structural computations increasing the possibility for better results in information retrieval. For example, speaking of query expansion[2] in search engines, given a query by the user, the structure server could traverse the synonymy relation for each appearing query term and replace it with its synonym. Similarly, in case the user wishes to broaden the search conceptually, the hyperonymy relation could be traversed adding the hyperonyms to the query, whereas for narrowing the query, the hyponymy relation should be traversed. Supporting this task within the CB-OHS framework provides the ground for delivering the computational tools to exploit wordnet structures.

[2] Query expansion is a method for performing sense-based retrieval by linking words based on their semantics [9].

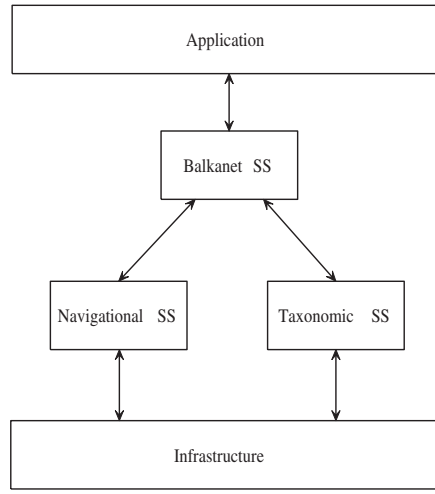


Fig. 2. The BalkaNet structure server and its incorporation in Callimachus system

4 Discussion

As it has been already mentioned, hypertext is not mainly used for the organization of information but can be considered as a significant means of structuring information. Viewing wordnets under the notion of hypertext, the power of the latter is enforced even more, making us infer that any kind of information can be structured under its fundamental characteristics. Taking advantage of the structural characteristics of hypertext, while developing wordnets, can prove quite beneficial for both the lexicographic and linguistic communities.

OHSs in their move from domain-specific frameworks to cross-domain CB-OHSs, provide the framework for supporting combinations among different domains. Based on the similarities that wordnets and already existing hypertext domains share, we came to the conclusion that even if wordnet is a new application domain cannot be seen as a new domain. In particular, the wordnet structure server relies on the basic structural abstractions of the navigational and the taxonomic domain, however, it supports a new set of behaviors providing the wordnet functionality.

The motivation of this work was the BalkaNet project whose application focuses on information retrieval issues. However, we strongly believe that by adopting structures implied by the hypertext community in other applications too, such as lexicography, the potential and performance of the latter can be greatly improved. When it comes to the storage of lexicographic data the need for efficient structures becomes apparent due to the large amount of information that has to be handled and especially due to the dynamic nature of the underlying information.

References

1. Bucka-Lassen, D., Pedersen, C. Å., and Reinert, Ó. H. 1998. *Cooperative Authoring Using Open Spatial Hypermedia*. M.S. Thesis, Aarhus University.
2. Bush, Vanevar. 1945. As we may think. *Atlantic Monthly* (Jul), 101-108.
3. Conklin, J. and Begeman, M. L. 1987. gIBIS: a hypertext tool for team design deliberation. *Proceedings of the 1987 ACM Conference on Hypertext* (Chapel Hill, NC), ACM Press, 247-251.
4. Conklin, J. 1987. Hypertext: An introduction and survey. *IEEE Computer 20*(9), 17-41.
5. Evens, M. W. (Ed.) 1988. *Relational models of the lexicon: Representing knowledge in semantic networks*, Cambridge University Press, Cambridge, England.
6. Fellbaum, C. (Ed.) 1998. *WordNet: An Electronic Lexical Database*, MIT Press.
7. Halasz, F. G. 1987. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Proceedings of the 1987 ACM Conference on Hypertext* (Chapel Hill, NC), ACM Press, 345-365.
8. Hasan, R. 1984. Coherence and cohesive harmony. *Understanding Reading Comprehension* (J. Flood, Ed.), IRA, Newark, DE.
9. Jing, Y. and Croft, W. B. 1994. An association thesaurus for information retrieval. *Proceedings of RIAO-94,* 4^{th} *International Conference "Recherche d'Information Assistee par Ordinateur"* (New York, NY), 146-160.
10. Kay, M. 1989. The concrete lexicon and the abstract dictionary. *Proceedings of the* 5^{th} *Annual Conference of the UW Center for the New Oxford English Dictionary* (Waterloo, ON), 35-41.
11. Kyriakopoulou, M., Avramidis, D., Vaitis, M., Tzagarakis, M., and Christodoulakis, D. 2001. Broadening structural computing systems towards hypermedia development. *Proceedings of the* 3^{rd} *International Workshop on Structural Computing* (Århus, Denmark), Springer Verlag, 131-140.
12. Landow, G. P. 1987. Relationally encoded links and the rhetoric of hypertext. *Proceedings of the 1987 ACM Conference on Hypertext* (Chapel Hill, NC), ACM Press, 331-343.
13. Leggett, J. J., and Schnase, J. L. 1994. Viewing Dexter with open eyes. *Communications of the ACM 37*(2), ACM Press, 76-84.
14. Lehmann, F. W. 1992. Semantic networks in artificial intelligence. *Semantic Networks* (F. W. Lehmann, Ed.), Pergamon Press Ltd., 1-50.
15. Marshall, C. C., Shipman, F. M., and Coombs, J. H. 1994. VIKI: Spatial hypertext supporting emergent structure. *Proceedings of the 1994 ACM European Conference on Hypermedia Technology* (Edinburgh, Scotland), ACM Press, 13-23.
16. Marshall, C. C. 1987. Exploring representation problems using hypertext. *roceedings of the 1987 ACM Conference on Hypertext* (Chapel Hill, NC), ACM Press, 253-268.
17. Millard, D. E., Moreau, L., Davis, H. C., and Reich, S. 2000. FOHM: A fundamental open hypertext model for investigating interoperability between hypertext domains. *Proceedings of the* 11^{th} *ACM Conference on Hypertext and Hypermedia* (San Antonio, TX), ACM Press, 93-102.
18. Miller, G. A. 1998. Nouns in WordNet. In [6], 23-46.
19. Nelson, T. H. 1974. *Computer Lib / Dream Machines*, Tempus Books of Microsoft Press.
20. Nürnberg, P. J., Leggett, J. J., and Wiil, U. K. 1998. An agenda for open hypermedia research. *Proceedings of the* 9^{th} *ACM Conference on Hypertext and Hypermedia* (Pittsburgh, PA), ACM Press, 198-206.

21. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As we should have thought. *Proceedings of the* 8^{th} *ACM Hypertext Conference* (Southampton, UK), 96-101. ACM Press.
22. Nürnberg, P. J., Leggett, J. J., Schneider, E. R., and Schnase, J. L. 1996. HOSS: A new paradigm for computing. *Proceedings of the 1996 ACM Hypertext Conference* (Washington, DC, Mar), ACM Press, 194-202.
23. Parunak, H. V. D. 1991. Don't link me in: Set based hypermedia for taxonomic reasoning. *Proceedings of the 3rd ACM Conference on Hypertext (Hypertext '91)* (San Antonio, TX), 233-242.
24. Priss, U. 1998. The formalization of WordNet by methods of relational concept analysis. In [6], 179-196.
25. Tzagarakis, M., Avramidis, D., Kyriakopoulou, M., schraefel, m. c., Vaitis, M., and Christodoulakis, D. 2002. Structuring primitives in the Callimachus component-based open hypermedia system. *Journal of Network and Computer Applications 25*(4)(Oct).
26. Wiil, U. K., and Nürnberg, P. J. 1999. Evolving hypermedia middleware services: Lessons and observations. *Proceedings of the 1999 ACM Symposium on Applied Computing* (San Antonio, TX), ACM Press, 427-436.

Software Development for Dynamic Systems

Darren Dalcher

Software Forensics Centre
Middlesex University
Trent Park, Bramley Road
London N14 4YZ, UK
d.dalcher@mdx.ac.uk

Abstract. The dynamic nature of knowledge and software evolution and usage present a pervasive challenge to system developers. Discrete attempts to create such systems often lead to a mismatch between system, expectation and a changing reality. The rationale for a Dynamic Feedback Model stems from the need to focus on a continuous and long-term perspective of development and growth in change-intensive environments. This paper makes the case for a reflective learning and knowledge-driven view of software development and presents such a model in a way that accounts for the long-term survival, growth and evolution of software-intensive systems.

1 Introduction

The starting point for exploring software engineering often revolves around the process view of software development and the implications and limitations that come with it. Software development is life cycle driven: the traditional linear life cycle, where progress is represented by a discrete series of transformations (state changes) has had a defining influence over the discipline. In fact, many software development processes are either predicated directly on various forms of a rational model concept or are designed to overcome perceived problems with this process through increments, timeboxes, user participation and experimentation (i.e. without "stepping away" from the problem).

Many of the life cycle variations appear to share a concern with software development as a one-off activity. The process is thus viewed as discrete projects or incremental sets of micro-projects that lead to a final integrated artefact. Indeed, the life cycle notion represents a path from origin to completion of a venture where divisions into phases and increments enable engineers to monitor, control and direct the activities in a disciplined, orderly and methodical way. The control perspective however retains a short-term perspective. The result is a limited emphasis on growth and improvement, and on the purchase of tools and investment in resource libraries. In practice, little attention is given: to global long-term considerations, to activities such as project management and configuration management that are continuous and on going, to the justification of the acquisition of tools and improvement strategies, to the need to maintain

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 58–75, 2003.

the value of the product and, more crucially, to the accumulation of knowledge, reflection, experience or wisdom.

2 Developing a Continuous Perspective

A life cycle identifies activities and events required to provide an idealised solution in the most cost-effective manner. No doubt many enacted life cycles are attempting an optimisation of determining a solution in "one pass", typically stretching from conception through to implementation, but others explicitly accept the unlikelihood of such a solution and are therefore iterative optimised over a single pass. The singular perspective can be represented as a black-box that takes in a given problem and resources as input and completes a transformation resulting in a delivered system that addresses the problem (see Fig. 1). The fundamental implications are: that the input consists of essentially perfect knowledge pertaining to the goal and starting state; that the transformation is the straightforward act of management and control within the established constraints; and, that the quality and correctness of the derived product is the direct output of that process. In this singular mode, knowledge is assumed to be fully available in a complete, well-understood, absolute and specifiable format. In order for the initial (static) abstractions to maintain their validity throughout the development effort, to ensure consistency and relevance, such projects need to be short enough (and possibly simple enough) so as not to trigger the need to update or discard the initial abstractions and thereby invalidate the starting baseline.

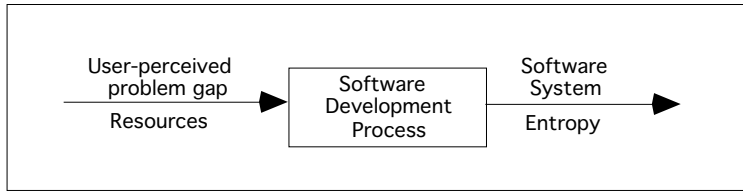


Fig. 1. The Singular Software Development Process

However, most real projects are not about well-understood problems that can be analysed and optimised, but are to do with knowledge that is elusive, tacit, incomplete, ephemeral and ambiguous. Working in ill-structured environments calls for continuous and adaptive design rather than complete and optimal analysis. The act of design is assumed to be a creative, experimental, argumentative and negotiation-based, discovery effort. Rather than being complete before inception, such situations require on-going elicitation of knowledge (requirements) and also require adaptability so as to respond to change. The primary output of requirements therefore is understanding, discovery, design, and learning, so that the effort is concept-driven [11,18,19].

Modern problem situations are characterised by high levels of uncertainty, ambiguity and ignorance requiring dynamic resolution approaches. Rather than expend a large proportion of work on the initial analysis of needs and expectations, such situations require openness to different domains and perceptions (rather than a search for optimality) and a recognition of the effect of time. As modern development focus moves from conventional commodities to a knowledge-based economy, intellectual capital becomes a primary resource. The main output of software development can be viewed as the accumulation of knowledge and skills embedded in the new emergent social understanding (see Fig. 2). Rather than act as input (cf. Fig. 1), knowledge emerges continuously from the process of development and learning. Continuous discovery and experimentation persist through the development, utilisation and maintenance phases. In addition to this primary output, the process may also create an artefact, which will represent a simplified, negotiated and constrained model of that understanding. Change combines with inputs of varying perspectives, and personal biases, attitudes, values and perceptions to offer the raw materials requiring constant compromise. The resolution process will thus result in a continuous output of understanding and insight that can replenish organisational skill and asset values. In change-rich environments, this output plays a far more crucial role to the future of the organism than the partial artefact whose construction fuelled the enquiry process. In the long run, the intellectual asset defines the range of skills, resilience, competencies and options that will become available.

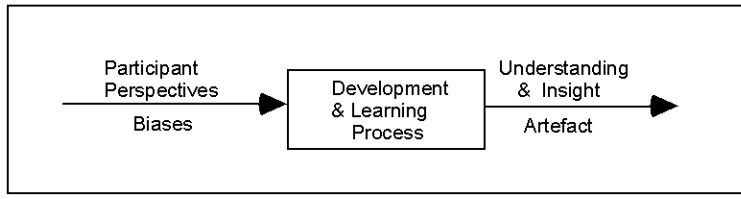


Fig. 2. The Development and Learning Process

The process of Problem Solving leads to improved understanding of a situation through discovery, experimentation and interaction with the problem domain. As the act of software design is increasingly concerned with the generation of understanding and knowledge, the main transformation needs to reflect the need for on-going exploration. A focus shift from delivering to discovering allows continuous exploration rather than temporal targets. Changing perceptions, needs, new learning and continuous experimentation will thus involve designers beyond discrete products and fixed specifications. Continuously evolving customer perspective and satisfaction provide the on-going success measures for determining achievement and utility. In a changing environment, quality is a moving target. Change and the lack of a permanent and static specification permit the view of customer satisfaction as an evolving level that is allowed to grow

and to improve alongside other assets. This stems from the recognition that satisfaction is neither a fixed nor a negotiated construct. The customer focus therefore extends beyond the product development view to incorporate usage and adaptation (of the product to the user rather than the other way round). Maintaining the evolving focus, value, relevance and satisfaction levels, requires a dynamic mechanism for conducting trade-offs as part of the sensemaking process. The process driver of knowledge-intensive development is the need to adjust to new discoveries and make trade-offs as part of the attempt to make sense.

3 Towards Continuous Software Engineering

Figure 3 emphasises some of the differences between the assumptions and the operating modes of different domains. Software engineering, as traditionally perceived, appears to relate to the resolution of Domain I problems. The domain of application for this type of approach is characterised by well-defined, predictable, repeatable and stable situations, reasonably devoid of change that can be specified in full due to a low level of inherent uncertainty. Progression is carried out sequentially from the fixed initial definition of the start-state to the envisaged final outcome. The main emphasis revolves around the transformation and fixed initial parameters to ensure that the end product is directly derivable from the initial state. Traditional approaches to quality assurance likewise depend on the belief that a sequence of structured transformations would lead to a high quality result. Repeatable and well-understood problems will thus benefit from this approach, assuming little scope for risks and surprises.

The middle ground (Domain II) is occupied by situations characterised by higher levels of uncertainty that cannot be fully specified. Despite a reasonable knowledge of the domain, there is still a degree of uncertainty as to the solution mode, making them more amenable to incremental resolution. Quality is also created and maintained incrementally but requires on-going effort (and organisation).

Far from being a structured, algorithmic approach, as is often advocated, software engineering in Domain III, is a continuous act of social discovery, experimentation and negotiation performed in a dynamic, change-ridden and evolving environment. Domain III systems are closely entwined with their environment as they tend to co-evolve with other systems, and the environment itself. Such systems affect the humans that operate within them (see for example, [21]) and the overall environment within which they survive, thereby, affecting themselves. This continuity acknowledges the evolving nature of reality (in terms of meanings and perceptions) and the fact that absolute and final closure in Domain III is not attainable. However, discovery and the concern with knowledge, enable a new focus on the long-term implications of development in terms of assets, improvement and growth. This view underpins the new emphasis on the growth of organisational resources, including knowledge, experience, skills and competencies, and on the improvement that it promises. (The shift therefore is not only from product to process, but also from discrete to continuous-and-adaptive.)

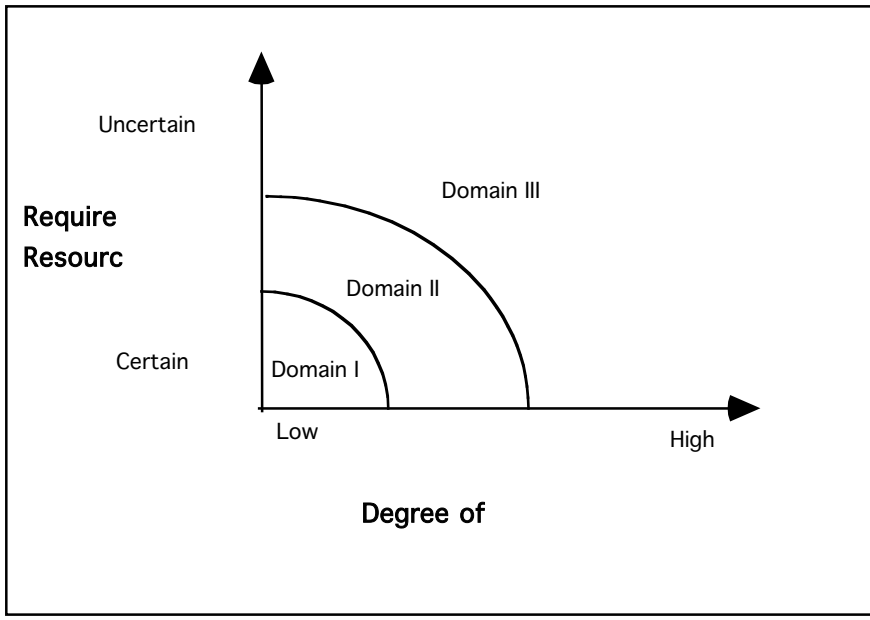


Fig. 3. Selecting a Life Cycle Approach

Generally, the temporal and discrete focus of software engineering for Domain I does not justify a long-term view. Rather than create assets for the long-run, software engineering is concerned with optimising new undertakings that are viewed as single and discrete projects and ignoring the long-term perspective. As a result, investment in quality and improvement is difficult to justify on a single project basis. Furthermore, no specific mechanisms are introduced to capture and record knowledge. Software engineering for Domain III should be concerned with the longer-term, where justification of resources extends beyond single projects. Investment in tools, improvement and new skills are therefore viewed against corporate targets of improvement and future performance levels which span multiple projects. Significant assets can thus be created in the form of knowledge, skills and competencies thereby serving the business and providing a competitive edge. Indeed, fostering a strategic view enables one to view all projects as strategic (as part of an asset portfolio). Maintenance becomes a part of normal protection, improvement and growth of assets, which also incorporates initial development.

A long-term perspective views design as on-going balance between concerns, constraints, challenges and opportunities. Problem solving in Domain III, involves continuous trade-offs in search of constraint satisfaction and opportunity maximisation. A unique feature of this perspective is the focus on the problem in terms of bi-directional adjustment of the starting position and expectations (rather than a uni-directional optimised process towards a static target). The

main implication is that users need to be involved throughout as their needs are likely to evolve as a result of the interaction between plan, action and reality. Customer interaction becomes on-going throughout the evolution of the system and the growth of the resource. Indeed, participation itself can be viewed as a growing asset which offers competitive advantages.

As highlighted above, the type of thinking required for Domain III situations relies on embracing a continuity dimension which enables long-term strategic thinking. The general conceptual shift therefore is from discrete software development (in Domain I) to problem solving in the continuous sense (in Domain III). A long-term perspective can account for knowledge through the required infrastructure supported by the adoption of the growing-asset perspective, thus underpinning normal development and evolution of values, assets, and satisfaction levels (assets in themselves). This allows for the fostering and improvement of expertise, skills and competencies offering a competitive edge through value-driven and experience-based improvement.

The proposed interpretation of growth and improvement is thus concerned with:

Intellectual Assets: Focused primarily on generating a strategic view of development that accounts for long term investment in, and growth of, intellectual assets.

Continuous Learning: Grounded in a systematic and multi-disciplinary perspective that is responsive to the results of continuous learning to enable such improvement.

Satisfaction and Utility Trade-offs: Concerned with the trade-offs and their implication on the resulting outputs and the emerging satisfaction and utility.

Monitoring Feedback: Based on constant monitoring and feedback which play a part in driving the process and directing the shifting of perceptions and values.

4 Models to Depict Development

Models are used as tools to aid in simplifying and reducing the complexity of reality by abstracting and focusing on certain essential portions of it. They are thus utilised in explanation, demonstration, description or prediction. One important distinction in modelling is that between static and dynamic models. Static models typically focus on the representation of states, while dynamic models concentrate on the representation of processes. Dynamic models have the power to depict processes as a continuous description of relationships, interactions and feedback. Feedback systems are critical to understanding relationships, interactions and impacts. They link causes and effects in dense and often circular causal patterns [7,24]. Feedbacks contribute to the ability to learn and improve and are therefore essential to understanding the growth of knowledge within structures and relationships. Information feedback plays a key part in connecting the different entities and in contributing to the growth and accumulation of knowledge resources. Utilising a model that uses feedback and is dynamic thus leads to a number of advantages:

Cross Discipline Perspective. First, an essential difficulty is in integrating knowledge from across disciplines in a way that supports growth and improvement. The traditional depiction of the process focuses on technical actions largely ignoring management processes and supporting activities. Indeed, while it is recognised that there are many interactions and interdependencies [12,13], little attention has been paid to how the different functions and domains are integrated [4,5]. Rather than being mutually exclusive, the different sets of activities support and impact across disciplines. An integrated model of the process would guide managers and developers towards a systems thinking approach to process modelling and away from the current focus on addressing a single aspect, or perspective of the process [7]. This type of shift would be difficult to adjust to, but the benefit of seeing a more complete picture and addressing the long-term concerns in a more informed way appears worth the additional investment in time and effort. An integrated view of processes serves to reaffirm the importance of major stakeholders involved in the enterprise of designing software, however indirectly associated with the act of software production. Indeed, such a model would be explicitly directed at the identified need for communication between disciplines [25] by attempting to create explicit links between the diverse domains.

Exploring Complexity. Second, the creation of extended systems takes in additional perspectives inherent in the composite of people, skills, and organisational infrastructure [20]. This enables the use of feedback to highlight and explore the complex interactions and loops that are formed within the software environment [1,24].

Viewing the Long-Term Horizon. Third, such a model enables reasoning about the long-term perspective. This facilitates the consideration of the long-term corporate objectives of enhancing quality, reducing maintenance, improving productivity, enhancing organisational assets and supporting growth which extend beyond the remit of single or discrete efforts. By looking beyond the delivered outputs it thus becomes possible to support the meeting of current and future objectives including the acquisition and maintenance of tools, skills, knowledge, co-operation, communication, double-loop feedback procedures and the competencies enabled by these combinations.

Wiser Trade-offs. Fourth, since, decisions are typically based on a single dimension or a single perspective view of the process prescribed by product-oriented or management activity models focusing on certain activities, a multi-perspective dynamic model can encourage intelligent trade-offs encompassing change and various concerns from the constituent disciplines. Furthermore, decision making and risk management assist in balancing long-term objectives and improvement with short-term pressures and deadlines. The model thus becomes essential to organisational decision making.

Dealing with Knowledge. Fifth, the model provides the mechanism for reasoning about knowledge and a framework for the long-term growth of knowledge and

assets. This enables reasoning about different perceptions and perspectives as well as encouraging a long-term approach to growth, improvement and support of organisational ability, knowledge, skills and competencies. The corporate perspective thus enabled, supports the growth and justification of resources that underpin an improving and evolving organisational asset.

Evolution and Continuous Learning. In addition, a dynamic model acknowledges the evolution of the system over time, thereby recognising the role of change and the need for continuous learning and adjustment (particularly prevalent in Domain III development). This goes to the core of the act of design, which involves learning and interaction with the various elements of the system in a constant effort to improve the fit in a rapidly changing (and often ill-structured) environment. It thus underpins the notions of a long-term perspective, evolution and growth.

5 The Dynamic Feedback Model

This section offers an example of a Dynamic Feedback Model (DFM) depicting the on-going inter-discplinary links and trade-offs embedded in development. Domain III software development projects embrace a complex set of interacting organisational entities that can normally be divided into three or four basic groups (encompassing functional domains). The relationships between the different groups can be understood by studying the feedback cycles operating between them. Modelling the relationships in a non-linear fashion allows a continuous view of the development effort to emerge, which in turn facilitates a long-term view of the management and dynamics involved in such an effort. The discussion that follows focuses on four different functional domains that are intertwined throughout the act of designing systems. Each perspective offers a valid representation of the attempt to develop a system. While they are often depicted as mutually exclusive, they tend to interact and influence other domains as they essentially refer to the same development undertaking [7]. This recognises the fact that design requires a more dynamic perspective that is responsive to changes, adaptations and complications, as the more traditional perspective is ill-equipped to handle dynamic complexity. Design thus entails trade-offs between perspectives and disciplines including management and quality of the required artefact.

The technical domain extends beyond the traditional notion of development to encompass the design and evolution of artefacts, which continues throughout their useful life span. Maintenance, from an asset perspective, is therefore part of this continuous cycle of improved value and utility. By extending the horizon of interest beyond the production stage, one becomes concerned with the useful life and persistence of artefacts, and in the continuous match between functionality, needs and environment. Knowledge is subject to growth and decay and enhances its value thorough sharing [17,26]. It features heavily within most disciplines and holds the key to success and prosperity through growth and the

reduction of entropy that have the potential to maintain it as the key resource in both ecologies and economies of scale and value. The continuous production of knowledge (viewed as development or design) thus facilitates the prospects of a long-term horizon [2,9,21]. The corresponding shift in perspective for software is from a product-centred focus to an asset-based perspective emphasising continuous usage, enhanced client emphasis and recognition of changing needs and perceptions.

The management domain is charged with the planning, control and management of the action that constitutes overall effort. It is therefore concerned with identifying mismatches, planning for action and assessing the degree of progress achieved, as well as with allocating resources to facilitate such progress. Trade-offs between levels of resources, attention and constraints will shape the resulting technical activity as the two are tightly bound. In the singular mode, management was perceived as a discrete and optimising activity guiding the transformation between the fixed input and the single output. Once the position of knowledge as an on-going commodity attained through on-going probing and discovery is acknowledged, and technical development is perceived as a long-term activity, management also takes on continuous dimensions [3,24]. Guiding exploration, adaptation and negotiation in search of a good fit, suggests a reduced role for absolute planning and a higher degree of responsiveness [8]. Discrete functional structure can therefore be dismantled in favour of knowledge or expertise driven enterprise (see for example, [27]). Furthermore, delivery is not limited to a single incident that only occurs once. The focus on delivering a product is thus replaced with the continuous need for generating and maintaining an on-going flow of knowledge requisite for the asset-based view of continuous development.

The quality domain is no longer optimised around the delivery of a product. The quality perspective in the singular "one-pass" perspective was assumed to be a derived attribute of the product emerging from the process. This notion also entailed correctness and implied acceptance by users. In ill-understood domains, quality is more difficult to capture and "fix". As discovery, learning and experimentation continue, and as adaptation rather than optimisation leads to a good fit, quality becomes an on-going concern. Experimentation and evolution lead to changes in expected functionality; and participation in experiments changes perceptions and expectations. Quality is thus perceived as a dynamic dimension, which continuously responds to perceived mismatches and opportunities reflected in the environment. It thus becomes more of a dynamic equilibrium [23], which requires constant adjustments and trade-offs. As development extends beyond the delivery stage, quality must also encompass the useful life beyond the release of an artefact. Satisfaction must cater to expectations and functional correctness leading to notions such as Total Customer Satisfaction [10] as the primary measurement of acceptability. In common with other domains, this perspective continues to evolve and grow, but can benefit from being viewed as an organisational resource or strength that needs to be maintained and enhanced.

The move from a target-based system towards an evolving fit entails a greater degree of uncertainty coupled with essential decision making and sensemaking.

Change, discovery and evolution also suggest a more open-ended environment seeking to balance and trade-off opportunities and risks. Risk management offers a domain for conducting argumentation, negotiation and trade-offs, while keeping the growth of the capital asset as a major objective. This enables skills and knowledge to benefit from the continuous search for fit, adaptation and evolution. User perceptions change partly due to discovery, but a discipline concerned with trade-offs can introduce a practical balance on the overall process, and in effect, focus not on the single transform box at the centre but act on all domains in a continuous fashion which was not possible in the more singular mode of problem solving. Not only does risk oversee all other domains including knowledge, it also directs the evolution by continuously balancing potential with constraints while enabling re-adjustment as a result of the accumulation of new knowledge. Risk management is therefore approached from a continuous perspective which attempts to allow for the distinction between reality and possibility by acknowledging the role of opportunity. Risk management thus offers "the glue" that unites the disciplines and concerns, thereby providing the capability for intelligent and "enlightened" trade-offs.

The discussion so far has established four general domains of interest (management, technical, quality and risk management). Generally, models fail to acknowledge the relationship between different domains such as management, engineering and product assurance primarily because they seem to emerge from one of the disciplines. The interactions between the different functions add to the overall complexity of the development effort and lead to emergent, yet unpredictable, behaviour dynamics [14]. This directly contributes to the need to focus on the ecology and interrelationships between the different components that make up the software environment. Figure 4 links the four domains of interest in a dynamic feedback loop.

Fig. 4. The Dynamic Development Feedback System

6 Feedback Loops

The dynamic model is in essence, a set of interactions and feedback loops governing and controlling the production of software from a continuous and dynamic perspective. It is also possible to analyse the model in terms of causal-loop diagrams. Space considerations prevent a detailed description of the loops which will only be briefly highlighted.

The basic loop in the dynamic system is the planning-control-visibility loop which enables the project manager to plan and control the production, evolution and growth of software by technical developers through the use of trade-offs and risk management (decision making). The loop enables the continuous generation of new information as well as feedback knowledge. The use of this knowledge is crucial in changing plans to adapt to reality and opportunities, modifying the decisions and re-examining the assumptions and the risk assessment completed previously. It also offers the framework for meeting scope, cost and interval targets. The recognition and establishment of visibility completes the basic feedback loop by providing a continuous process to ensure the system remains relevant with regard to its objectives.

Some form of product (or service) assurance is required in order to store, assess, and evaluate completed baselines at milestones and analyse the effectiveness of the processes producing them. In addition, the planning and controlling activities operate in concert with the risk management function. This is provided by the configuration control loop, linking control, baselines and monitoring that enables the (closed-loop) link between monitoring and controlling. Monitoring provides feedback on the strategies implemented, thus enabling more effective control. Monitoring relies on quality assurance techniques and feedback mechanisms to evaluate progress and quality. Monitoring is the missing link between risk management and quality as it makes the feedback system a closed-loop system. In this way, monitoring enables learning from past mistakes, improvement, the gradual accumulation of knowledge and the growth of competencies.

Reporting and evaluation are not represented in typical life cycle models or in decision making processes. Their role is to provide the feedback (and long-term knowledge) that allows plans and decisions to be revised. Such revisions pertain to integrated technical plans as well as to objectives and strategic targets, in which case they can play a part in adjusting management plans. This is depicted by the reporting-planning loop.

7 Evaluation

The DFM depicts relationships between the different organisational entities and can be used as a framework for understanding the dynamic nature of the interactions between entities in Domain III development. This model breaks away from linear thinking and the temporal project perspective to offer a continuous perspective for understanding and implementing the relationships and their long-term effects. This is done through the depiction of on-going feedbacks and

interactions. The model therefore offers an alternative method of reasoning about development. A single project perspective ignores the dynamics (see for example, [14]) that are established between the functions as it assumes that the espoused linear perspective and the organisational forms that accompany it can adequately capture the nature of the "ideal" environment. Such "short-termism" obstructs the nature and critical importance of any interactions other than the prescribed linear ones. A long-term perspective enables developers to view the process and the organised structure as a framework for capturing knowledge and regarding it as an asset. This can lead to gradual continuous improvement.

The DFM highlights the fact that some consideration must be given to non-technical aspects of the development effort. Indeed, the purpose of the central subsystem is to integrate different perspectives and considerations and make sense of the emergent knowledge, thereby enabling intelligent decision making and balanced trade-offs. The establishment of four major domains suggests that effort must go into recognising and organising all four systems. Many organisations are characterised by defective management strategies, ineffective quality assurance and an immature process (at the initial level as defined by the SEI). Intelligent deployment depends on improvement to these areas, which will thrive in the context of holistic (if somewhat contention-inducing) decision making. In fact, creative contention exercised on a continuous basis, may result in richer understanding, wider choice options and greater utilisation of possibilities leading to increased prosperity.

The driving process is essentially, a simple feedback web that offers great diversity through complex interactions. The recognition of the process as a dynamic and on-going activity seems to lead to improved management perception which in turn, may result in more adequate planning and resource allocation. Better understanding of the process and the available knowledge deals with the essence of the problem, offering the long-term potential for reduction of associated problems such as maintenance (which can be viewed as an integral part of on-going value enhancing of corporate assets). The inherent simplicity of the model (which is an inherent characteristic of complex dynamics – see for example, [15,22]) makes it an attractive and workable formalism for process improvement. Moreover, the long-term perspective shifts attention from supporting the product to supporting the process, thereby enabling a longer-term improvement through investment in tools (and at a higher level in infra-structure, and in tools for meta-development).

8 The General Model

Software development, like any other form of design or engineering is described as a decision making process (see for example, [9]). The objective of such a process is to make a rational choice from among a set of alternatives in an attempt to maximise the expected value of the consequences on the basis of available knowledge, preferences, desires, expectations and opportunities [8].

We can relabel the DFM diagram to highlight the generic essence and the role of the model in addressing issues of decision making and focusing on the dynamic discovery and generation of knowledge. The framework for organisational decision making requires a decision making engine at the root, with systematic procedures enforcing the effective operation of the engine. The engine interacts with other functional organs via a well-established set of control mechanisms, carefully balanced by the engine itself. Knowledge generated through the technical domain is used by the engine during the decision making process. Knowledge acquired from implementing earlier decisions can then be fed-back into the process either as background knowledge available to support future decisions, or as a formalised part of an integral body of knowledge which can be used to optimise the decision making process. Such a framework needs to be dynamic and feedback-oriented in order to maximise and improve its ability to make optimal decisions. A representation of such a process is depicted in Fig. 5.

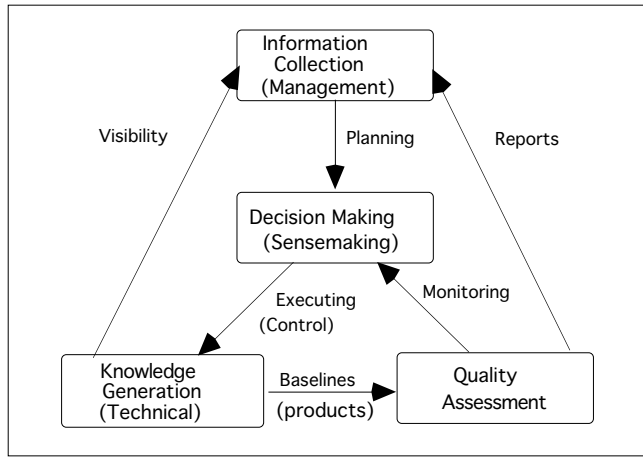


Fig. 5. The Generic Dynamic Knowledge Feedback Model

The decision making subsystem is the engine running the entire system. Procedures operating inside this sub-system result in the establishment of good decisions (with respect to available knowledge). Arriving at rational decisions may be vital to survival but the ultimate goal must be the implementation of these decisions and the monitoring of their execution on a continuous basis. A complex mechanism is used by the engine to control this process and guide its interactions with the other major sub-systems. Effective decision making requires educated utilisation of existing information. This information is provided by the planning activity which is the organisational framework for activities and decision making. The information is fed into the decision making engine to ultimately deliver decisions. Decisions affect the entire system in general, but relate to the knowledge generating activity in particular. The technical activity that

produced the information is supported by all other mechanisms but essentially it is the design function that represents the purpose of the entire production effort. Similarly, at a more fundamental level, this activity is concerned with generating knowledge (taking the economic perspective of knowledge as both a growing organisational asset and the essence of production).

The combination of planning, controlling and monitoring, bridges the gap between the current state and the desired state and enables the act of design (i.e. it represents the management of change which underpins long-term evolution). The executing activity directs and controls the operation of knowledge generation by implementing the decisions that trigger on (and offer limits and constraints on) knowledge production. Continuous assessments of decisions (including assumptions and rationale) and their consequences provides the on-going monitoring baseline. The purpose of monitoring is to establish the effectiveness of the decision, to suggest corrective measures, and to utilise this information to improve the dynamic decision making process itself, and the overall knowledge resource.

Controlling the execution of decisions, which generate new knowledge, results in released artefacts as well as the generation of new knowledge. The process of generating artefacts and achieving targets is visible to the management subsystem in its capacity for information collection. The quality assessment function continuously measures and assesses the products against pre-defined standards and transmits the results to management to generate an additional knowledge-flow source. Some of the information is also used to initiate the monitoring and improvement activity with respect to the decision making process and forthcoming continuing execution tasks. Reports and visible information are collated by the information collection sub-system and used to initiate systematic re-planning of tasks. Essentially, the general model is therefore concerned with the continuous generation and utilisation of knowledge as part of the discovery and adaptation that typify evolving designs.

Decision making is an attempt to balance knowledge, uncertainty, ambiguity and ignorance with a view to maximising the expected returns on an on-going basis. The creation of artefacts is led by decisions and is responsible for generating more information and insights (as a result of failures) into the process. In fact, sensemaking and decision making offer the ability to deal with ambiguity, risk and ignorance to enable the design. For an ideal operation, this information needs to be assessed, collated and reused to result in more efficient decision making processes and enhanced skills.

The generic model is predicated on this objective and shows that the creation of artefacts relies on the continuous generation and handling of knowledge. The four major entities deal with some form of acquired knowledge which ranges from products (an abstract form of knowledge) to specific reports, and from skills to organisational competencies and resources. The activities which connect decision making to other major subsystems are also intrinsically knowledge intensive. They depict the relationships and interactions between the different functions and decision making, as well as revealing their continuity and system-

wide relevance in terms of feedbacks and the knowledge they convey. Indeed, Niwa [16] pointed out that risk management acts as a knowledge transfer system which is concerned with collecting experience, providing adaptability and ensuring improved ability.

The central sub-system offers the rationale for the existence of the entire system and therefore co-ordinates and holds together the other elements. Most decision making mechanisms include some implicit consideration of risk but software development efforts with their inherent problems (and troublesome history and track record) must incorporate explicit risk assessment, mitigation and containment functions. In fact, most of the traditional process models rely on a simplified, and overly simplistic form of cost/benefit analysis which neglects all consideration of risks.

The execution of agreed strategies results in the generation of knowledge which can be viewed, assessed and altered to improve future performance. The quality assessment entity generates additional information about past decisions, products and current processes, while the information collection function attempts to centralise all available knowledge and rearrange it in a form which will be useful for decision making. Overall therefore, efficient knowledge work leads to more achievements and to better chances of utilising opportunities.

9 Summary and Discussion

This paper focused on the on-going aspects of Domain III software development by introducing the DFM that encompasses the main functions concerned with design. Dynamic models rely on feedback which enables focus on the continuous nature of processes and their long-term survival, growth and evolution. The model highlights the need for continuous discovery of knowledge which essentially drives the feedback interactions leading to growth. The focus on knowledge as a key asset can be transported to software development to support the view of continuous growth and improvement as an essential corporate asset.

The DFM encourages thinking about software development in terms of the constituent parts and their interactions. The model addresses another fundamental deficiency in contemporary processes by offering a decision making perspective. This view of the process in terms of problem solving offers visibility into the mechanisms required to deal with the inherent uncertainties embedded in the environment. Most contemporary models miss this aspect of decision making [5] and subsequently cannot adequately address the hidden aspects of uncertainty, ambiguity, risk and conflicting objectives. The DFM uses risk management and the control framework driving it to direct the entire process. By taking a long-term perspective, it becomes possible to reason beyond the time horizon or resources of a single project to obtain an enlightened insight and structure that can improve performance and become a valued and growing corporate resource in the long-term.

The DFM is receptive to changes in the environment (characteristic of Domain III) and tackles them by feeding acquired knowledge back into the decision

making subsystem. Thus, if, for example, a prototyping strategy is employed to reduce the inherent risk by buying information, the information can be used to improve the overall process and the knowledge resource. This approach to risk-containment and adjustment to new knowledge, can be adapted to any situation. The interest in risk enables integration with any risk-driven approach and supports management in appreciating the organisational relationships, dynamic interactions and dependencies which affect and direct such risk assessments.

Activities such as configuration management are phase-independent and cannot be fitted into an individual stage in more traditional life cycles. Many such activities are not integrated into conventional models, as they represent a longer-term perspective and outlive the traditional process and are consequently considered to reside outside the remit of development. The DFM on the other hand, justifies the long-term investment in supporting facilities.

The model provides a unique possibility for accounting for and reasoning about long-term decisions such as creating a reusable software library, investing in tools or information, or accepting new and innovative processes. Moreover, the DFM seems to account for evolution, growth, change and maintenance (i.e. there is no need to re-organise the structure, change the functions, or create a new model). The software process can thus be perceived as a continuous growth of software assets over time. A corresponding improvement in shared skills and competencies across projects is also supported.

As multiple views of the process are integrated, a more complete picture of the effort can be constructed. However, as the business of software development becomes more integrated in management practices and ever-growing organisational contexts, and as software becomes more integrated in other areas, the importance of continuous learning, knowledge, and skill acquisition as underpinned by this model will remain central to improved control, visibility and management. Therein lies the value of models such as the DFM: the availability of the model coupled with risk management and control principles, and adaptive frameworks for growth and knowledge could serve as a stepping-stone on the way to establishing an overall holistic and strategic project management framework, thereby supporting development as a continuous activity subject to long-term improvement.

Moreover, as software development becomes embedded in larger and more comprehensive processes, it becomes a more global concern that extends beyond the remit of software engineering and impacts the organisation at a number of levels. A Domain I short-term perspective appears incapable of supporting and addressing these aspects. A Domain III long-term perspective can account for knowledge through the required infra-structure supported by the adoption of the growing asset perspective, thus underpinning normal development and evolution of values, assets, and satisfaction levels (assets in themselves). This allows for the fostering and improvement of expertise, skills and competencies offering a competitive edge through value-driven and experience-based improvement.

Rather than bring into focus the accepted facets of software engineering process, this paper reflects a different direction – one that considers a longer-term

focus, while bringing interactions, feedbacks and accumulation of competencies into play. An important aspect is its focus on continuity and growth from a long-term perspective, beyond the normal remit of development, where the accumulation of intellectual assets represents continuous improvement in organisational capability and potential.

This notion appears to represent a distinct departure from the traditional Domain I focus on discrete and limited efforts viewed in isolation. It also re-emphasises human issues and attention to social and political constraints, issues and processes. Furthermore, a long-term view justifies the adoption of multiple perspectives, the reuse of knowledge and the utilisation of a dynamic perspective which underpin feedback and improvement. Taken together, this points to a move from a structured engineering to a growing design perspective; a move from discrete and project-based Domain I software engineering to a strategic, continuous and evolving Domain III discipline of software design.

This interpretation is not proposed as a final answer, but as an intermediary step – an opening of a new perspective on long-term development and improvement.

References

1. Abdel-Hamid, T., Madnick, S. E. 1991. *Software Project Dynamics: An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ.
2. Alexander, C. et al. 1977. *A Pattern Language*, Oxford University Press, Oxford.
3. Cleland, D. I. 1998. Strategic planning. *Field Guide to Project Management* (D. I. Cleland, Ed.), Van Nostrand Reinhold, New York, 3-12.
4. Curtis, B., Krasner, H., Shen, V., Iscoe, N. 1987. On building software process models under the lamppost. *Proceeding 9th International Conference on Software Engineering* IEEE Computer Society, 96-103.
5. Curtis, B., Krasner, H., Iscoe, N. 1988. A field study of the software design process for large systems *Communications of the ACM 31*(11), 1268-1287.
6. Dalcher, D. 2000. Feedback, Planning and Control – A Dynamic Relationship. *FEAST 2000*, Imperial College, London, 34-38.
7. Dalcher, D. 2001. Life cycle design and management. *Project Management Pathways: A Practitioner's Guide* (M. Stevens et al., Eds.), APM Press, High Wycombe.
8. Dalcher, D. 2002. Safety, risk and danger: a new dynamic perspective. *Cutter IT Journal 15*(2), 23-27.
9. Dym, C. L., and Little, P. 2000. *Engineering Design: A Project Based Introduction*, John Wiley, New York.
10. Ireland, L. R. 1998. Total customer satisfaction. *Field Guide to Project Management* (D. I. Cleland, Ed.), Van Nostrand Reinhold, New York, 351-359.
11. Lawrence, B. 1998. Designers must do the modelling. *IEEE Software 15*(2) (Mar), 30-33.
12. Lehman, M. M., Belady, L. A. 1985. *Program Evolution: Processes of Software Change*, Academic Press, London.
13. Lehman, M. M. 1998. Software's future: Managing evolution. *IEEE Software 15*(1), 40-44.

14. Lehman, M. M. 2000. Rules and tools for software evolution planning and management. *FEAST 2000*, Imperial College, London, 53-68.
15. Marion, R. 1999. The Edge of Organisation: Chaos and Complexity Theories of Formal Social Systems. Sage, London. (1999).
16. Niwa, K. 1989. *Knowledge-based Risk Management in Engineering*, John Wiley, New York.
17. Plotkin, H. 1994. *The Nature of knowledge: Concerning Adaptations, Instinct, and the Evolution of Intelligence*, Allen Lane, London.
18. Reifer, D. J. 2000. Requirements management: the search for Nirvana. *IEEE Software 17*(3) (May), 45-47.
19. Sackman, R. B. 2000. Overview for success in the Internet age. *PM Network 14*(6), 54-59.
20. Senge, P. M. 1990. *The Fifth Discipline*, Doubleday, New York.
21. Simon, H. A. 1996. *Sciences of the Artificial* (3rd ed.), MIT Press, Cambridge, MA.
22. Stacey, R. D. 1992. *Managing Chaos: Dynamic Business Strategies in an Unpredictable World*, Kogan Page, London.
23. Turner, J. R. 1993. *The Handbook of Project-Based Management*, McGraw-Hill, London.
24. Weick, K. E. 1979. *The Social Psychology of Organising* (2nd ed.), Addison Wesley, Reading, MA.
25. Yeh, R. T., Naumann J. D., Mittermir, R. T. 1991. A commonsense management model *IEEE Software 8*(6), 23-33.
26. Young, J. Z. 1987. *Philosophy and the Brain*, Oxford University Press, Oxford.
27. Zohar, D. 1997. *Rewiring the Corporate Brain: Using the New Science to Rethink How we Structure and Lead Organisations*, Berret Koehler, San Francisco.

Containment Modeling of Content Management Systems

Dorrit H. Gordon and E. James Whitehead Jr.

Department of Computer Science
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, California 95064
{dgordon, ejw}@soe.ucsc.edu

Abstract. Containment models are a specialized form of entity-relationship model in which the only allowed form of relationship is a 'contains' relationship. This paper builds on the authors' previous work on containment modeling by clarifying the graphical notation and increasing the expressiveness of the model in a way that expands the technique to systems outside the hypertext domain. Examples from the hypertext, hypertext versioning, and configuration management domains illustrate the cross-domain applicability of this technique.

1 Introduction

The past 35 years have witnessed the emergence of a broad class of systems to manage digital content. Under the banners of Hypertext, Software Configuration Management (SCM), Document Management, VLSI Computer Aided Design (CAD), and Media Asset Management, theses systems have been developed to address the distinctive requirements of each community. Despite the many differences among the systems in these domains, a current of commonality runs through them all: each of these systems provide a repository that contains all content, and within this repository content can have associated metadata, and can be aggregated together into collections. These systems commonly offer change control functionality: facilities for versioning content, collections, and sets of these. The systems provide multi-person write access and hence have capabilities for group work, ranging from per-object concurrency control to multi-object collaborative workspaces.

To date, it has been difficult to explore the commonality among these disparate systems since there is no easy way to compare their data models. In our previous work, we introduced a novel modeling technique that focuses on expressing the containment relationships between entities in hypertext systems [10,11]. This *containment modeling* technique has been used to describe the data models of a broad range of hypertext systems, including monolithic (open) hyperbase, linkbase, and hypertext versioning [14], as well as the Dexter reference model [6] and the Web with WebDAV extensions [12]. When applied to hypertext systems, containment modeling provides several benefits, including the ability

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 76–89, 2003.

to concisely represent complex data models, and easily cross-compare the data models of multiple systems. It provided a data modeling language that could be used to describe and reason about sets of hypertext systems, rather than single systems.

In this paper, we expand the descriptive scope of containment data modeling to encompass both Software Configuration Management systems, and Aquanet, the first spatial hypertext system. Aquanet represents a class of hypertext systems not previously modeled. In the process of modeling these systems, we make multiple refinements to containment data modeling to adequately capture the nuances of each system. Together, these refinements form the primary contribution of this work: extending the scope of containment modeling to handle systems outside the hypertext domain. In so doing, we demonstrate that containment data modeling is cross-domain, capable of modeling content management systems in general, not just hypertext systems. Furthermore, by modeling multiple SCM, hypertext, and hypertext versioning systems in one place, in a consistent notation, we are able to perform a cross-domain comparison of the data models of these systems.

In the remainder of the paper, we provide a description of the enhanced containment data modeling technique, including the semantics of containment, and a graphical notation for their visual representation. We validate containment data modeling by using it to model five hypertext systems representative of the monolithic, spatial, and hyperbase systems, along with three SCM systems, and one hypertext versioning system. Together, these models highlight the containment model's ability to represent complex data models of a range of content management systems. After each set of models, the paper discusses observed similarities and differences among the systems. We conclude with a brief look at future directions for this research.

2 Containment Modeling

Containment modeling is a specialized form of entity-relationship modeling wherein the model is composed of two primitives: entities and relationships. Each primitive has a number of properties associated with it. A complete containment model identifies a value for each property of each primitive. Unlike general entity-relationship models, the type of relationships is not open-ended, but is instead restricted only to varying types of referential and inclusive containment.

2.1 Entity Properties

Entities represent significant abstractions in the data models of content management systems. For example, when modeling hypertext systems, entities represent abstractions such as works, anchors, and links, while in configuration management systems, they represent abstractions such as versioned objects and workspaces. The properties of entities are:

Entity Type. Entities may be either containers or atomic objects. Any entity which can contain other entities is a container. Any entity which cannot container other entities is an atomic object. Any entity which is contained in a container may be referred to as a containee of that container.

Container Type. There are two sub-types of container: 'and' and 'xor'. Unless otherwise specified, a container is generic (i.e. neither 'and' nor 'xor'). 'And' containers must contain two or more possible containee types and must contain at least one representative of each; 'xor' containers must have two or more possible containee types and may contain representatives of no more than one of them.

Total Containees. A range describing the total number of containees that may belong to a container. A lower bound of zero indicates that the container may be empty.

Total Containers. A range describing the total number of containers to which this entity may belong. A lower bound of zero indicates that the entity is not required to belong to any of the containers defined in the model. Note that for models in this paper, it is assumed that all elements reside in some type of file system, database, or other structure external to the system being modeled.

Cycles. Many systems allow containers to contain other containers of the same type. For example, in a standard UNIX filesystem, directories can contain other directories. This property indicates whether a container can contain itself (either directly or by a sequence of relationships). Unless otherwise specified this paper assumes that cycles are allowed (where existing containment relationships make them possible).

Ordering. The ordering property indicates whether there is an order between different containee types (ordering of multiple containees of the same type is captured as a relationship property). For example, a full name contains a first name, zero or more middle names, and a last name. The ordering property on the full name container indicates that the first name goes before the middle names which go before the last name. The importance of the ordering of the middle names with respect to one another is captured by the ordering property on the relationship connecting full name to middle name.

Constraints. The constraints property allows us to express other restrictions on the containment properties of an entity. Two kinds of constraints currently identified are mutual exclusion and mutual inclusion.

A mutual exclusion constraint on a container indicates that it may contain one or the other, but not both, of a pair of possible containees. This occurs in configuration management systems where a special entity (the versioned object) exists to hold versions of an object. In this case, an instance of a versioned object holds versions of only one object type, although there may be several objects which are versioned in this way.

A mutual inclusion constraint on a container indicates that it must contain both (or neither) of a pair of possible containees. For example, in a system which maintains metadata about its contents, it might be possible to create an empty repository, but it would be impossible to create an item in the repository with-

out creating metadata and it would be impossible to create metadata without creating an item.

These constraints have been described as they apply to containers. They may also apply to entities with respect to their containers. That is, there may be a mutual exclusion constraint on an entity which indicates that if it belongs to one container, it may not belong to another.

The mutual inclusion constraint can also be broken down so that it is unidirectional. For example, a graph can contain nodes without containing edges, but it cannot contain edges without containing nodes.

2.2 Relationship Properties

Containment Type. Containment may be either inclusive or referential. Where inclusive containment exists, the containee is physically stored within the spaced allocated for the container. This occurs, for example, when an abstraction is used to represent actual content (as opposed to structure) in the system. Referential containment indicates that the containee is stored independently of the container.

Reference Location. The references which connect containers and their containees are usually stored on the container; but on some occasions they are stored on the containee, or even on both container and containee. This property indicates where the references are stored.

Membership. The membership property indicates how many instances of a particular container type an entity may belong to. If the lower bound on the value is zero, then the entity is permitted, but not required, to belong to the container indicated by the relationship.

Cardinality. The cardinality property indicates how many instances of a particular containee a container may contain. If the lower bound on the value is zero, then the container is permitted, but not, required, to contain instances of the containee indicated by the relationship.

Ordering. The ordering property indicates whether there is an ordering between instances of a particular entity type within the container indicated by the relationship. FOR example, full names contain zero or more middle names. Middle names are ordered in the full name container. Mary Jane Elizabeth Smith is not equivalent to Mary Elizabeth Jane Smith. We could imagine entering everyone's middle names into a drawing for a prize. The collection of entries (assuming no one's cheating) would be unordered.

Containee Type. This property is unique to versioning systems. It distinguishes between relationships where the container contains multiple versions of a single containee, single versions of multiple containees, or multiple versions of multiple containees. This only becomes important when the cardinality of the relationship is greater than one. It is important because many configuration management systems have examples of two or more of these cases.

As an example of the first case, some configuration management systems use an abstraction of an object to contain all the version of that object. There might

be many objects in the system, but each abstraction contains version of only one object.

As an example of the second case, configuration management systems all have some mechanism for selecting the specific version of a number of objects to include in a release.

As an example of the third case, some configuration management systems allow users to construct spaces with any contents they desire (potentially including multiple objects and multiple versions of those objects).

2.3 Graphical Notation

Entities and Their Properties. Entities are represented as nodes of the containment model graph. The following legend shows the representation of entity properties on the graph.

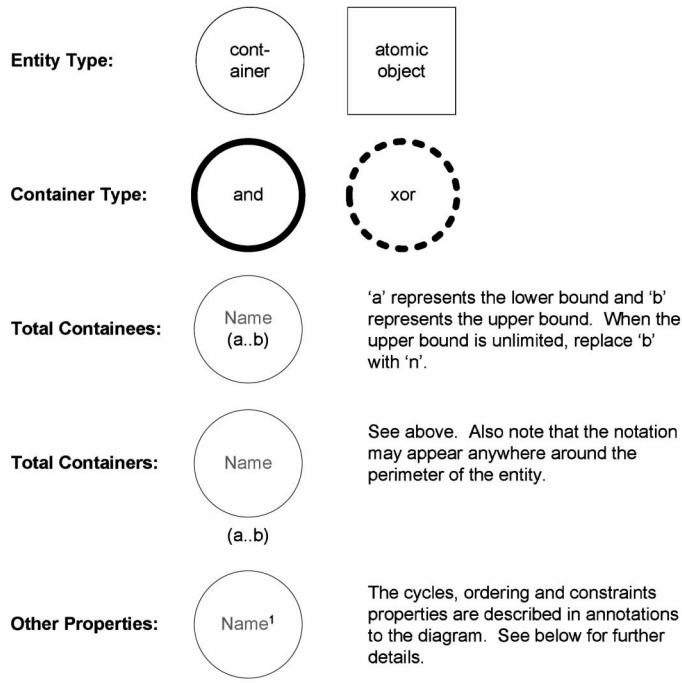


Fig. 1. Entity Legend

Unless otherwise specified, it is assumed that, where possible, cycles are permitted. If cycles are not permitted, the statement *no_cycles* appears in a container's annotations.

If there is an ordering among different containee types in a container, the ordering is described by an ordering statement. Ordering statements have the

following syntax: *order*(containee 1, containee 2, ..., containee n). The containees are listed in order. Only those containees involved in a partial ordering are listed. If there are multiple partial orderings, the annotation contains one ordering statement per partial ordering.

Constraints are defined using predicate logic. The most common predicate is *contains*(container, containee). A mutual exclusion constraint would appear as follows: *contains*(container, containee 1) $\leftrightarrow \neg$*contains*(container, containee 2).

Relationships and Their Properties. Relationships are indicated by directed edges between pairs of entity nodes in the containment model graph. In the following legend, visual elements that represent the source end of an edge are shown on the left ends of lines. Visual elements that represent the sink end of an edge appear at the right ends of lines.

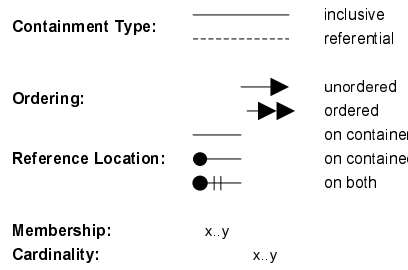


Fig. 2. Relationship Legend

In models of versioning systems the containee type is just in front of the cardinality, followed by a colon. A 'V' is used for version, an 'O' for object, and a 'VO' for both. For example, if the container may contain one or more versions of a containee, the containee type and cardinality would be noted on the diagram as, "V:1..n."

Why Not UML? It is possible to draw a containment model using the standard Unified Modeling Language (UML) notation, so why have we chosen to develop a different notation? There are two reasons: readability and utility. Figure 3 shows two different versions of the same simple containment model: one drawn using UML, the other drawn with the notation presented in this paper. The UML diagram has considerably more elements than the diagram that uses our notation, because relationship properties beyond cardinality and membership must be described using UML's annotation mechanism. Applied to a more complex model, UML notation produces a cluttered, overly complex diagram that is difficult to read, and which may be too large to print or display conveniently.

The UML representation is also less useful than a representation using the notation we have developed. By describing as many properties as possible graph-

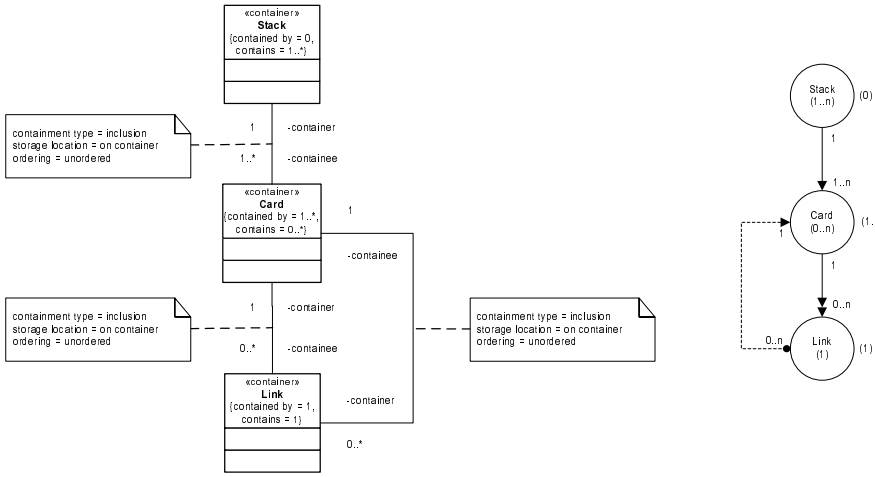


Fig. 3. UML and non-UML representation of the HyperCard [1] containment model

ically, we enable the user to quickly and easily compare models of different systems. The visual representation allows users to quickly and easily spot the similarities and differences between systems. In the UML notation, a user would need to laboriously compare the annotations on each relationship to determine whether patterns existed.

Although we do not believe the standard UML notation is sufficient for containment modeling, it would be possible to frame the notation described here as a UML extension. The stereotyping mechanism in UML allows the creation of new types of objects and allows the assignment of unique visual representations for those types.

2.4 Changes to Containment Modeling

In this paper, we have improved upon the containment modeling technique previously described in [11], in two ways. First, the graphical representation now shows each axis of variability in the model as a single axis of variability in the diagram. For example, ordering is shown by having either one arrowhead (unordered) or two (ordered); previously, a circle at the tail of a containment relationship showed ordering. This frees the circle to be used exclusively to represent the location of the reference identifier in the referential containment.

Second, we chose to eliminate inheritance and storage relationships from the modeling system in [11]. Inheritance was only used occasionally, and then only to reduce clutter on diagrams. When appropriate, it can be represented by a separate inheritance hierarchy, such as a UML structure diagram. Storage relationships were mainly used to relate the system to the external structure (such as a file system) where it resided. A gain, this may be valuable information, but is not truly a part of the containment structure of the system. We have simplified

the containment modeling technique so that it is limited to relationships that are entirely within the system.

2.5 A Brief Note about Cycles

Upon initial examination, it may seem that simply identifying any container as allowing or disallowing cycles would be sufficient. In some circumstances, however, it is not. For example with ClearCase [7], as in many file systems, directories may contain other directories. This relationship may be cyclic; that is, a directory may, by reference contain another directory. There is nothing to prevent the second directory from also containing the first. On the other hand, some SCM systems express versioning by creating a linear ordering of objects, where each object points to (contains by reference) its successor (or predecessor). This requires the self-referential containment relationship to by acyclic. For SCM systems which version directories this produces a contradiction. The self-containment relationship which indicates that one directory can contain another may be cyclic. The self-containment relationship which represents the linear ordering of versions must be acyclic. Our current cycle property will not accommodate this situation. We leave to future work the development of a richer cycle property (or set of properties) to allow us to correctly model this case.

3 Containment Model Examples

3.1 Hypertext and Hyperbase Systems

Containment modeling allows us to see at a glance the similarities and differences between various systems in a domain. An examination of Fig. 4 shows that all hypertext systems center around three main constructs: collections or groups of objects, objects, and connections between objects. We have all used the Web and recognize these general characteristics of hypertext systems. Containment modeling illustrated them clearly.

It also enables us to recognize differences. A key difference between hypertext systems lies in their treatment of links. The Web and HyperCard both embed a set of links in the content of each object as with the "<A href=" construct of HTML. many hypertext systems, however, do not use embedded links: NoteCards, Aquanet, and HyperDisco, for example. Among these systems, though, there is a variation in what the links contain. Links in NoteCards and Aquanet referentially point to a content object, while in HyperDisco, links contain lists of "to" and "from" endpoints. An endpoint is an abstraction that contains an anchor and either a node or a composite. While not a complete survey of data models for linking, it seems reasonable that, having modeled the containment structures of multiple hypertext systems, we could analyze these structures to develop a design space of anchoring and linking.

In addition to comparing analogous features of a system, containment modeling can show us overall system characteristics. For example, the containment

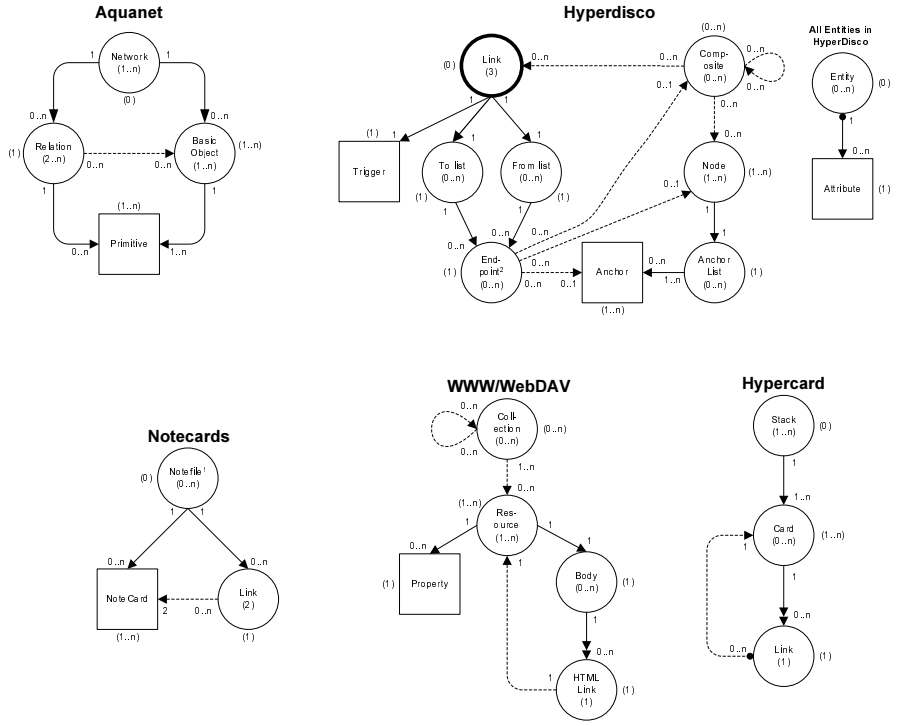


Fig. 4. Containment models of selected hypertext systems: Aquanet [8], HyperCard [1], NoteCards [9], WWW/WebDAV [12], and HyperDisco [13]. Constraints on the diagram are 1 (NoteCards): *contains*(Notefile, Link) → *contains*(Notefile, NoteCard), and 2 (HyperDisco): *contains*(Endpoint, Node) ↔ ¬*contains*(Endpoint, Composite) and *contains*(Endpoint, Node) ⋁ *contains*(Endpoint, Composite) → *contains*(Endpoint, Anchor)

models in Fig. 4 show that both HyperDisco and WWW/WebDAV maintain some metadata. The HyperDisco metadata system is quite extensive. An entity in the system may have attributes associated with it. In addition, links have a particular piece of metadata, a trigger, associated with them. The WWW/WebDAV metadata is less extensive. It is limited to properties on resources. None of the other systems show evidence of metadata collection in their containment models.

As with any model, we must recognize that the information drawn from containment models has its limitations. For example the containment models for Aquanet, HyperCard, and NoteCards show no evidence of metadata. That does not mean that these systems cannot support metadata collection. Any of them could, at a policy level, be used to maintain metadata simply by requiring anyone who adds a new resource to the system to link it to a metadata resource. Similarly, while the containment models for WWW/WebDAV and HyperDisco both illustrate explicit metadata systems, neither system can force users to apply

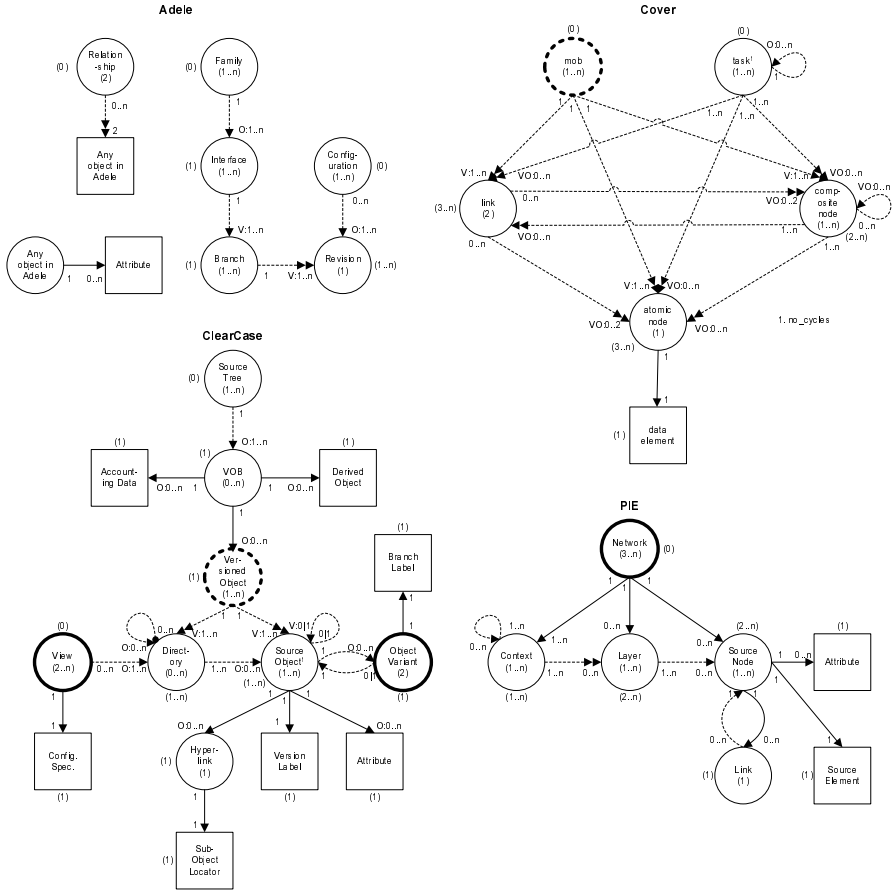


Fig. 5. Containment models of selected configuration management (Adele [2], ClearCase [7], and PIE [3]) and hypertext versioning systems (CoVer [4,5])

metadata appropriately. For example, keyword searches on the Web often produce totally irrelevant information — because website owners frequently pack their site's keywords with irrelevant words to generate more hits from search engines.

3.2 Configuration Management Systems

As we saw with hypertext systems, containment modeling allows us to look for similarities and differences among systems within a domain.

By examining the models of SCM systems, we can see a fundamental structural difference between PIE and the others. Both ClearCase and Adele depend on collecting all versions of a given item into a single container that represents the abstract notion of each item, independent of a specific revision or variant. In

ClearCase this container is the "versioned object," while in Adele it is the "interface," which contains a set of branches representing versions and variants of that interface. The versioned object abstraction is a key enabler for automated assistance in constructing configurations. It allows the system to recognize when two different items in the system represent the same abstraction at different points in time (versions) or space (variants). Versions are recognized because they belong to the same versioned objects. Management of varian and Adele. In ClearCase, variants are explicitly tagged with metadata and are contained within a versioned object. In Adele variants belong to different branches, but to the same interface.

PIE has no analogous structure. This key difference gives PIE users extraordinary flexibility in how they can manipulate and organize their environments, but at the expense of automated support for establishing and maintaining configurations. The more rigorous structure in the other SCM systems provides them with the information necessary to automatically select items to include in a configuration based on higher-level parameters set by the users.

3.3 Cross-Domain Comparisons

In addition to intra-domain analysis, containment modeling can be used to compare systems across domains. We can examine SCM systems and hypertext systems to identify patterns that are common to both. We can also readily identify elements that distinguish the two kinds of systems.

The containment model of CoVer is interesting because it combines aspects of both hypertext systems and SCM systems. Across the center of the model we see a core structure that is common to hypertext systems: a collection of objects and the links that connect them.

At the top and bottom of the model we see structures that are similar to those found in SCM systems. The mob container holds all the versions of a particular object, analogous to the versioned object container in ClearCase. The task container may hold any combination of versions and objects. It fills the workspace niche (called a 'view' in ClearCase).

The SCM containment models show us that designers of SCM systems have also drawn from the hypertext world. All three systems modeled here have link structures. The links even exhibit a similar range of traits to the links we saw in hypertext systems. Some are embedded within specific kinds of entities (as in ClearCase and PIE), and others are independent (as in Adele). In none of the systems are links treated as independently versionable objects.

CoVer is unusual in its treatment of links. We saw examples of independent links in both hypertext and configuration management systems. The independent link entity in CoVer is unusual because it is versioned independently. None of the SCM systems treat links as explicitly versionable. They are versionable in ClearCase and PIE only because they are embedded in other entities that are versioned. Adele does no allow versioning of links at all.

We have seen how containment modeling can highlight similarities between systems in different domains. It can also help determine what makes the domains

different. We saw that both hypertext and SCM systems have collections of nodes and the links between them. Hypertext systems stop there. They have three main elements: collections, nodes, and links. While some hypertext systems have more than three entities in their models, the additional entities are either parts of the link implementation, or are entities that represent actual data or metadata. SCM systems, on the other hand, all have at least on more layer. Each SCM system has at least one entity that collects collections. In fact, these meta-collections are roughly parallel to the hypertext collections. The lower level collections (those that collect nodes) are the versioning mechanisms. In ClearCase and Adele the role of these collections is clear: they contain the individual versions of a single, conceptual entity. In PIE and CoVer their role is less explicit since these low-level collections can actually contain any nodes, irrespective of the nodes' relationships to one another. In all four systems, however, these low-level collections are, themselves, collected to construct a complete picture of the system's content.

4 Conclusion

Each system within the SCM and hypertext domains can be viewed as an exploration of a specific point within a multi-faceted space of design choices. Many SCM and hypertext systems have been developed over the past 30 years, providing the raw data that can be analyzed to develop a description of the key decision points in the common design space of content management systems.

The containment modeling technique presented in this paper permits the data models of content management systems to be graphically represented and compared to reveal their commonalities and differences. Containment modeling was applied to a set of nine hypertext, SCM, and hypertext versioning systems, the first time such a broad set of these systems has been compared. Containment modeling can thus be viewed as a meta-modeling technique, in that it provides a mechanism capable of representing the data models of systems in multiple domains.

We intend to use this work as the basis for a thorough domain analysis of content management systems. Containment modeling will allow us to investigate systems in the Hypertext, SCM, Document Management, and VLSI CAD domains, and to identify the key structural components of each. It will help us to distinguish between truly different structures and those that differ only marginally, or only due to terminological differences.

Beyond modeling, we intend to develop a system that can automatically generate content management systems. Given a precise, machine-readable description of a containment model the generator will automatically create a repository consistent with that specification. Automatic generation will provide future researchers and systems builders with a powerful tool for the rapid creation of content management repositories, and the exploration of more complex and powerful data models.

Furthermore, by using the same tool to automatically generate a broad range of content management systems, it will reinforce the commonalities among content management systems.

Acknowledgments

We would like to thank Jim Jones for his generous editing assistance, and Uffe Wiil for clarification of the HyperDisco containment model. Additionally, we thank Jessica Rubart for her detailed thoughts on how to express containment modeling in UML (not all of which were explored in this paper), David Millard for his feedback from using containment modeling to represent the data model of the Open Hypermedia Protocol, and GuoZheng Ge for his detailed review of the PIE containment model. Ongoing work in containment modeling is supported by National Science Foundation grant CAREER CCR-0133991.

References

1. Apple Computer. 1988. *HyperCard Script Language Guide*, Addison-Wesley, Reading, MA.
2. Estublier, J. and Casallas, R. 1994 The Adele Configuration Manager. *Configuration Management* (W. F. Tichy, Ed.) John Wiley & Sons, New York, 99-133.
3. Goldstein, I. P. and Bobrow, D. P. 1984 A layered approach to software design. *Interactive Programming Environments* (D. R. Barstow, H. E. Shrobe, and E. Sandewall, Eds.) McGraw-Hill, New York, NY, 387-413.
4. Haake, A. 1994. Under CoVer: the implementation of a contextual version server for hypertext applications. *Proc. Sixth ACM Conference on Hypertext (ECHT'94)* (Edinburgh, Scotland, Sep), 81-93.
5. Haake, A. and Haake, J. 1993. Take CoVer: Exploiting version support in cooperative systems. *Proc. InterCHI'93 - Human Factors in Computer Systems* (Amsterdam, Netherlands, Apr), 406-413.
6. Halasz, F. and Schwartz, M. 1994. The Dexter Hypertext Reference Model. *Communications of the ACM 37*(2), 30-39.
7. Leblang, D. 1994. The CM Challenge: Configuration management that works. *Configuration Management*,(W. F. Tichy, Ed.) John Wiley & Sons, Chicester, 1-38.
8. Marshall, C. C. Halasz, F. G. Rogers, R. A. and Janssen, Jr., W. C. 1991. Aquanet: a hypertext tool to hold your knowledge in place. *Proc. Third ACM Conference on Hypertext (Hypertext'91)* (San Antonio, TX, Dec), 261-275.
9. Trigg, R., Suchman, L., and Halasz, F. 1986. Supporting collaboration in NoteCards. *Proc. Computer Supported Cooperative Work (CSCW'86)* (Austin, TX, Dec), 147-153.
10. Whitehead, Jr., E. J. 2001. Design spaces for link and structure versioning. *Proc. Hypertext 2001, The Twelfth ACM Conference on Hypertext and Hypermedia* (Århus, Denmark, Aug), 195-205.
11. Whitehead, Jr., E. J. 2002. Uniform comparison of data models using containment modeling. *Proc. Hypertext 2002, The Thirteenth ACM Conference on Hypertext and Hypermedia* (College Park, MD, Jun), 182-191.

12. Whitehead, Jr., E. J. and Goland, Y. Y. 1999. WebDAV: a network protocol for remote collaborative authoring on the Web. *Proc. Sixth European Conference on Computer Supported Cooperative Work* (Copenhagen, Denmark, Sep), 291-310.
13. Wiil, U. K. and Leggett, J. J. 1996. The HyperDisco approach to open hypermedia systems. *Proc. Seventh ACM Conference on Hypertext (Hypertext'96)* (Washington, DC, Mar), 140-148.
14. Østerbye, K. and Wiil, U. K. 1996. The Flag Taxonomy of open hypermedia systems. *Proc. Seventh ACM Conference on Hypertext (Hypertext'96)* (Washington, DC, Mar), 129-139.

Smart Active Object: A New Object-Oriented Programming Paradigm for Developing Multithreaded Applications*

Yan Chen, Xinyuan Fan, Jian Jiao, and Dongmin Wang

Dept. of Electronics Engineering
Shanghai Jiaotong University, 200030 Shanghai, China
{chenyan, fxylily, jiaojian, dongmin_wang}@sjtu.edu.cn

Abstract. Multithreaded programming has been widely used nowadays, but developing mutlithreaded applications is still a tough task. To alleviate such kind of work we propose a new concept, *Smart Active Object*, an extension to the normal object in object-oriented technology. *Smart Active Object* runs concurrently with the main execution logic of the program, leveraging multithreading mechanisms and meanwhile leaves only sequential interfaces for programmers to insert application-specific parts, masking details of multithreading, which eventually enhances programming productivity. To support our claims, we design and implement a framework comprising a collection of cooperative *Smart Active Objects* for stateful applications, a typical problem in telecom world. Using this framework, we easily solve a toy problem, derived from the behavior of SIP stateful proxy servers within much shorter time than a conventional C version counterpart.

1 Introduction

In the past few years multithreaded programming [5] has been widely used in a wide variety of application domains including Graphical User Interface(GUI), scientific computing, and network services [1,9,2,4]. People prefer it to conventional programming styles because: it is easy to share data among different threads of control, it increases application throughput and responsiveness, it is less time and resource consuming than maintaining processes, and it can take advantage of parallel hardware architectures, such as SMP.

Attractive as the above features are, multithreaded programming is still a tough task for programmers to take. It is difficult to choose the right programming paradigm: whether to use request per thread, thread pool, or thread group whose size expands and shrinks over the time. It is difficult to synchronize: which data shall be protected, and where to put the mutex. It is difficult to track memory leak when memory are newed and deleted frequently, passing control among different threads. It is difficult to debug as current debuggers are still not mature enough for thread management. It is difficult to estimate performance before the

* This research is supported by Ericsson.

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 90–99, 2003.

application is developed and put to field testing. All these make multithreaded programming a burdensome, time-consuming but still error-prone approach to developing real applications.

Current available multithreaded programming interfaces are not that satisfying. POSIX thread, Win32 threads and Solaris threads [5] are all general solutions, exhibiting all the previously discussed shortages. OpenMP [6,7], a proposed standard for scientific computing, seems useless when an application is not filled up with intensive computing loops.

In this paper, we propose a new concept, *Smart Active Object*(SAO), to alleviate this programming complexity. SAO is object, which exhibits all the features of normal objects, such as encapsulation, inheritance, and etc. It is also active, which means it may run concurrently during the program execution through multithreading mechanism. Further, it is smart, which means programmers can use it without caring about multithreading details, and only have to specify application-dependent parts in a sequential way.

This paper is organized as follows. In Sect. 2, we describe the origin for SAO. In Sect. 3, we show the background for stateful applications and demonstrate a commonly used mutlithreaded programming paradigm. In Sect. 4, we discuss in detail about our framework comprising of a collection of SAOs for stateful applications. In Sect. 5, we present a toy problem solved by this framework to support our claims. Section 6 concludes the paper.

2 Smart Active Object

With the introduction of multithreaded programming we can add one more feature to Object-Oriented(OO) technologies: changing passive objects into active ones. Consider a toy string class, which only has two public member functions, "add", attaching a string to the object and "show", printing the string of that object. After defining such an object in a program, programmers can use it by calling the above two functions. And programmers are sure that until the next call of "add", all the calls of "show" will give the same result. That is to say: such kind of objects are passive because they refuse to work unless you call their member functions explicitly in your program. Here, our modification is: making the object active by leaving them run in another thread of control, i.e., a distinguished thread other than the main thread. With this, programmers can get various results by calling "show" even between two adjacent calls of "add".

Though the new concept is interesting and fun, it alone is not enough to alleviate programmers's work. Because to make it of practical use, all the thread management details have to be considered by programmers themselves, such as inserting mutex to protect shared variables. Java standards [3] has included thread as one of its classes. It shares some idea of active objects as ours', but still leaving all the work about thread management, to programmers. In this respect, such kind of solution shows nothing advanced from normal multithreading interfaces.

So we need a kind of *Smart Active Objects*, which can:

- present the same programming experience as normal objects.
- mask all the details of multithreaded programming.

Still take the toy string class as example. Suppose the toy string class has a fancy character: it deletes the first character of the string it maintains every second. We say such kind of objects are SAO, if programmers can still use the member function calls, "add" and "show" as usual, and the results output by the object are logically correct sticking to the specified functionality, and there is no additional thread management calls in the application codes.

We argue that SAO is a possible solution because problems of a type often share some general multithreading paradigm. For example, a common paradigm for network services is On-Demand Invocation [10,12], i.e., spawning a new thread for each arrival of requests; a usual mechanism for a thread to pass messages to another existing thread is through use of condition variables, a notion described in POSIX Thread.

If a complex problem involves a collection of cooperative SAOs, a framework can be designed, assembling these objects together and masking more details such as how these SAOs contact with each other, which further alleviates programmers' work.

In the following sections we describe a type of common problems in telecom world, stateful applications and elaborate our object-oriented framework for it, leveraging the concept of SAO together with other C++ features.

3 Stateful Applications and Their Common Multithreaded Solution

Stateful applications are a common phenomena in telecom systems, such as switches and call control servers. In the scenario of plain telephone services, a state machine will be assigned to each call attempt for establishing, monitoring and terminating the call session. As many people may have talks simultaneously, a switch must be ready to maintain a large amount of such state machines.

Multithreaded programming maps well with such kind of problems. Each message received by the switch must be directed to the corresponding state machine. Such kind of message passing can be quite easy if the message receiver and state machine maintainers are threads within the same address space. The concurrency properties of threads can keep each state machine run simultaneously, which eventually presents a high throughput image to customers.

A general multithreaded solution for this sort of stateful applications comprises three types of threads. One thread, say A, is responsible for getting messages from the outside world. After receiving a message, it creates a new type of thread, say B, to process the message while it itself again rolls back waiting for other messages. B, after getting the message, makes a decision on which existing state machine shall receive the message. If B cannot find one, it creates another kind of thread, say C, to set up a new state machine to process the message,

or just do some otherwise functions specified by the switch if for example the message does not happen to be a valid one; if somehow B finds one, it just forwards the message. After that B is terminated. C maintains a state machine, which is triggered by both incoming messages and internal timers. When the state machine comes to an end at last, C terminates but before that it must clear all the information about its existence so that later messages would not be routed to this non-existing thread.

To make the system operate correctly, mutex and condition variables must be applied for synchronization among threads. One mutex must be used to protect the consistency of all the existing call state machines. Each state machine must have a mutex for the consistency of its pending messages, and a condition variable, providing a mechanism for B notifying C of an incoming message.

4 Framework

4.1 Abstract Model

According to the previous discussion, we identify three fundamental parts to construct a stateful system. They are, *Reception*, accepting incoming messages and forwarding to other modules, *Dispatcher*, distributing inbounding messages to the right state machine, and *State_Keeper*, shepherding state machines. Figure 1 illustrates this abstract system model.

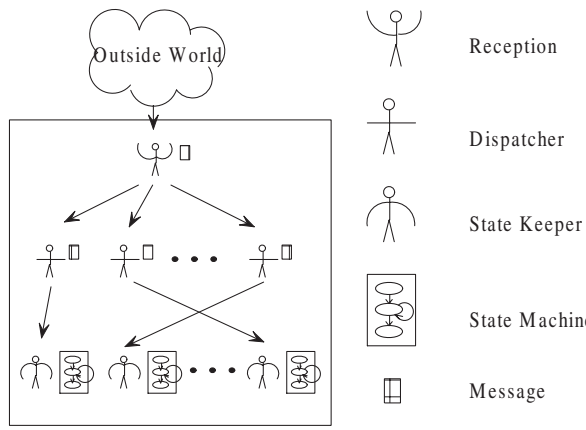


Fig. 1. Abstract model for stateful systems

Our solution in this paper is to map each instance of a module to an SAO, hiding multithreading details and presenting application-specific interfaces in a sequential manner. As these SAOs cooperate with each other we combine them into a framework.

For the current implementation, we design three base classes, class Reception, class Dispatcher, and class State_Keeper corresponding to the three modules we describe previously. All these classes are defined as *templates*, a feature provided in C++ as we want the framework to mask how these modules contact with each other. For example, Reception is parameterized by a derived class from Dispatcher so that it can pass incoming messages to Dispatcher transparently. Another reason for using templates is because we want to mask management details of internal messages, which is often error-prone if controlled by programmers themselves. As the format of the internal message cannot be known beforehand, all these three base classes have it as a parameterized type. In addition another C++ feature, *virtual functions* are used to present unified interfaces for programmers to insert application-specific parts.

A common interface function throughout these classes is member function "activate" which makes the object run. Essentially it just creates a thread, running another member function which contains the actual service logic for that object. Through this interface, for example, Reception can create and activate a new Dispatcher without knowing how it runs.

We then in the remaining sections discuss our implementation details for each module in turn.

4.2 Reception

Functionality. This module is the entrance point for the whole system. It fetches raw data from the outside world and parses it into a well-formed message for Dispatchers.

Things that can be masked:

1. Thread management details.
2. The mechanism for contacting Dispatcher module.
3. The iterative behavior of accepting incoming messages.

Application-specific parts in this module:

1. The details about fetching data from the outside world and parsing it into application specific message format.

Class Description. Public member function, "fill_out_msg" is the interface for programmers to specify how to fetch and parse incoming data into a well-formed internal message. It is declared as virtual function for the derived class's more precise definition.

Private member function, "event_loop" is the main servic logic for *Reception*. First it news an empty internal message for programmers to fill in. After a message is filled out, the message is forwarded to a new Dispatcher for further delivering while the main thread itself just rolls back for more messages. Two nice features in this function are: 1) It invokes a new Dispatcher module by just passing a message's pointer without knowing how that module runs in the background. 2) It offloads work of newing space for each incoming message, which if handled by programmers may lead to some memory leak.

4.3 Dispatcher

Functionality. Dispatcher module is in charge of distributing an incoming message to the right State_Keeper and if failing to find one, it creates a new state machine or does some otherwise operations. It is also Dispatcher's responsibility to keep track of all active State_Keepers.

Things that can be masked:

1. Thread management details.
2. The mechanism for contacting with State_Keeper Module.
3. The details of keeping track of active State_Keeper modules.

Application-specific parts include:

1. The details about getting the state machine id from the message content.
2. The rule for deciding whether to create a new state machine or do otherwise operations based on the message content if failing to find any matching state machine.
3. The actions to take if failing to find any matching state machine but not necessary to create a new state machine.
4. The method to free the memory occupied by the message content.

The reason for adding 4) as one of the interface function because we wish the Dispatcher can automatically destroy the message after processing if necessary.

Class Description. Four virtual functions, "get_mach_id", "is_nece_create", "do_otherwise" and "free_msg_content" are the interface functions corresponding to the four application-specific parts mentioned previously respectively.

Private function, "dispatch_process" is the main service logic for this module. It uses the same algorithm as we describe in Sect. 3 for thread B. If a new state machine needs to be created, Dispatcher just news and activates a State_Keeper and calls State_Keeper's member function, "add_msg" for further process. If a matching State_Keeper is found, a call of "add_msg" can also forward the message to the State_Keeper. With this interface function call to State_Keeper module, Dispatcher is not at all necessary to know the underlying mechanisms for message passing between Dispatcher and State_Keeper. That is SAO's attribute.

Private variable, "store" is a structure used to manage all the active State_Keepers, with "state_record_queue" storing records for active State_Keeper, and "state_record_queue_mutex" protecting its consistency.

Another important interface function is "delete_state_record". This interface function is used for a State_Keeper instance to delete its record in the dispatcher before killing itself so that further messages would not be routed to the non-existing SAO. It gets the mutex, "state_record_queue_mutex" in "store" before deleting the corresponding record for consistency.

4.4 State_Keeper

Functionality. State_Keeper's main function is to monitor the behavior of the state machine. It accepts incoming messages and timers and then takes up appropriate actions. In addition, it must notify Dispatcher about its termination before ending.

Things that can be masked:

1. Thread management details.
2. The strategy for managing pending messages.
3. The mechanism for handling incoming messages and timers simultaneously.
4. The mechanism for notifying Dispatcher module about its ending.

In this module, application-specific parts include:

1. Actions to take upon receipt of timers and messages.
2. Strategy for freeing the memory occupied by the message content.

Class Description. Virtual functions, "timer_callback" and "msg_callback" are interface functions for programmers to insert application specific actions on timers and messages.

Private function, "event_loop" is the main service logic for this object. In this implementation, all the incoming messages are arranged in a FILO queue, "msg_queue", which is protected by a mutex, "msg_queue_mutex". The condition variable, "msg_queue_cond" makes it possible for the thread to wait for incoming messages and pending timers. After going out of event loop in case of an ending, it calls Dispatcher's static function "delete_state_record" to inform Dispatcher about its ending. Two nice features exists in this module: 1) State_Keeper module does not at all need to know how Dispatcher module keeps track of active State_Keepers. 2) Programmers do not have to care about freeing the internal messages, as all these tiresome work are taken by our framework.

Some assistance functions are provided for use in "timer_callback" and "msg_callback" functions. They are: "get_state", getting the current state of the state machine, "set_state", setting new state for the state machine, "set_timer", set new timer, and "end", indicating the end of a state machine.

The public member function, "add_msg" is the interface function for Dispatcher module for directing messages. It essentially use the condition variable's mechanism for signalling State_Keeper of this information.

4.5 A Simple Program Paradigm

To develop stateful applications on top of our framework, programmers only have to define three derived classes from Reception, Dispatcher, and State_Keeper respectively. In the simplest situation, only four member functions have to be redefined. They are: 1) Reception::fill_out_msg, 2) Dispatcher::get_mach_id, 3) State_Keeper::timer_callback, 4) State_Keeper::msg_callback.

A simple program paradigm is show in Fig. 2. It can be noticed that no multithreading codes exist and every function is realized in a sequential way.

```
//include base classes
#include "reception.h"
#include "dispatcher.h"
#include "statekeeper.h"

//internal message format
typedef struct _Message {
  ...
} Message;

//declaration of derived classes
class Toy_Dispatcher;
class Toy_State_Keeper;

//****************************
//Toy Reception module
//****************************
class Toy_Reception : public
  Reception<Message, Toy_Dispatcher> {
public:
  Toy_Reception()  {
    //prepare passive socket
    sock = socket(...);
    ...
    bind(sock, ...);
  }

  int fill_out_msg(Message &msg)  {
    //receive data from outside world
    n = recvfrom(sock, buf, ...);
    ...
    parse(buf, msg);
    ...
  }

private:
  void parse(char *buf, Message &msg) {...}
  int sock;
}; //end of class Toy_Reception

//****************************
//Toy Dispatcher Module
//****************************
class Toy_Dispatcher : public
   Dispatcher<Message, Toy_State_Keeper> {
public:
  Toy_Dispatcher(Message *msg) :
       Dispatcher<Message, Toy_State_Keeper>(msg) {};
  void get_mach_id(const Message &msg, char *id)  {...}
}; //end of class Toy_Dispatcher

//****************************
//Toy State Keeper Module
//****************************
class Toy_State_Keeper :
       public State_Keeper<Message, Toy_Dispatcher> {
public:
  enum {STATE1, STATE2, ...};

  void timer_callback(void)  {
    ...
    //take propriate actions according to current state
    state = get_state();
    switch(state)    {
    case STATE1:
      ...
    case STATE2:
      ...
    }
  }

  void msg_callback(Message &msg) {
    ...
    //take propriate actions according
    //to current state
    state = get_state();
    switch(state)    {
    case STATE1:
      ...
    case STATE2:
      ...
    }
  }
};//end of class Toy_State_Keeper

//****************************
//main
//****************************
int main()
{
  Toy_Reception *recep;

  recep = new Toy_Reception;
  recep->activate();
  pause();
}
```

Fig. 2. A simple program paradigm developed on our framework for stateful applications

5 Example

5.1 Our Toy Problem

SIP [8] stateful proxy servers are typical stateful systems in telecom domains. A famous multithreaded implementation for SIP stateful proxy server is SIPD [11], which has been the template for our toy problem. We simplify the operation of the proxy server in both running mechanism and message format.

Our system comprises two kinds of components: client and server. The client side issues calls to the server. The call establishment between the two sides is a simplified version of SIP "invite" transaction. Both sides keep state machines throughout the call transaction. But only two kinds of responses are implemented, the temporal response, 180 and the final response, 200.

5.2 Programming Experience

Some programming experience we gain during the development is:

– We spent only a man-day finishing the whole application, comparing to a seven man-day work for a C version one.

– We hardly used GDB for debugging but some printing functions helping us find faults in service logic rather than thread management logic. But in the development of the C version one, we spent most of the time in debugging, tracking bugs in both service logic and thread management logic.
– Our C++ version has about 550 lines of code, including some comments. And the C version one reaches 800 lines of code.

5.3 Measurements

Our testbed includes two parts: four client nodes and a server node. In each client node, we set up a group of client state machines, which simultaneously issued calls to the server. We varied the number of client state machines in each node from 100 to 250. Throughout our test, we recorded the time span for each call establishment attempt. Figure 3 shows the distribution of the call establishment time span under different workload. The call ratio is calculated in the unit of five seconds, starting from the 15th second.

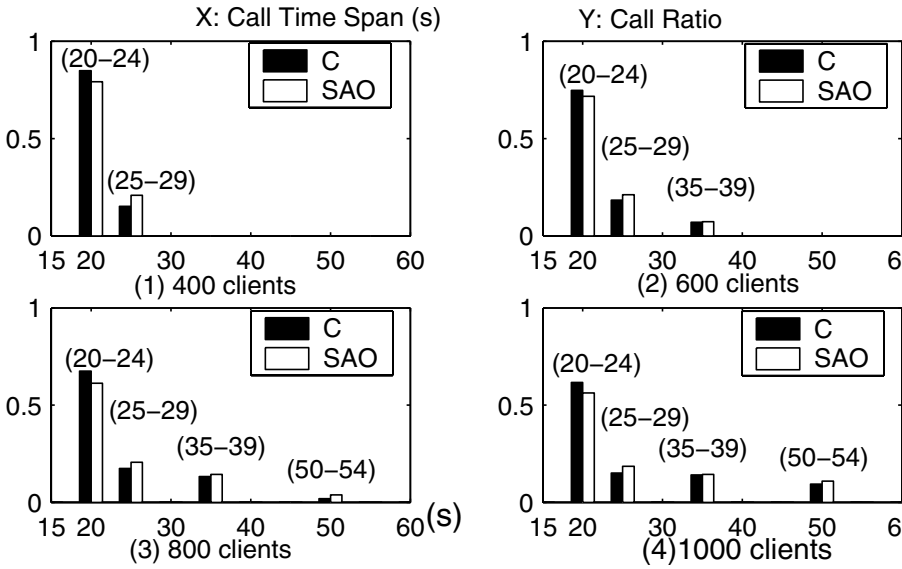


Fig. 3. Performance comparison under different workload

It can be seen that most calls can be finished between the 20th and the 24th second as we set up a minimal timer up to 21 seconds within the call establishment protocol. With the growth of workload, all the calls can still be delivered but with an increase of delayed call ratio. Our framework solution has only minor performance degradation compared to its C version counterpart under all these workloads, proving its working efficiency.

6 Conclusions

In this paper, we introduce a new concept, *Smart Active Object* to alleviate multithreaded programming complexity. Equipped with this new concept, we design an object-oriented framework for stateful applications and conduct a toy problem to show its practicality. Our conclusion is that: when programmers develop multithreaded applications on top of a framework comprising a collection of cooperative SAOs, it can greatly improve their programming productivity with performance assured.

Acknowledgements

We would like to thank anonymous reviewers for their valuable comments. Special thanks are also given to Mr. Hans Brolin from Ericsson for discussing telecom type applications with us.

References

1. Apache HTTP server project. `http://httpd.apache.org/`.
2. Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A., and Gauthier, P. 1997. Cluster-based scalable network services. *Proc. of the 16th ACM Symposium on Operating Systems Principles* (Oct).
3. Gosling, J., Joy, B., Steele, G., and Bracha, G. The Java Language Specification (Second Edition), `http://java.sun.com/`.
4. Holmedahl, V., Smith, B., and Yang, T. 1998. Cooperative caching of dynamic content on a distributed web server. *Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing* (Jul).
5. Lewis, B. and Berg, D. J. 1996. *Pthread Primer- A Guide to Multithread Programming*, Sunsoft Press, ISBN 0-13-443698-9.
6. OpenMP Architecture Review Board. OpenMP: a proposed industry standard API for shared memory programming. White paper. `http://www.openmp.org`.
7. OpenMP Architecture Review Board. OpenMP C and C++ Application Program Interface(version 1.0), `http://www.openmp.org`.
8. RFC2543. SIP: Session Initiation Protocol.
9. Saito, Y., Bershad, B. N., and Levy, H. M. 1999. Manageability, availability, and performance in Porcupine: a highly scalable, cluster-based mail service. *Proc. of the 17th ACM Symposium on Operating Systems Principles* (Dec).
10. Schmidt, D. C. 1994. A domain analysis of network daemon design dimensions. *C++ Report 6* (Mar).
11. SIPD. SIP proxy, redirect and registrar server, `http://www.cs.columbia.edu/IRT/cinema/`.
12. Stevens, W. R. 1998. *UNIX Network Programming(Vol.1, 2nd Edition): Networking APIs: Sockets and XTI*, Prentice Hall, ISBN 7-302-02942-3.

Implications of the View of Programs as Formal Systems

Bertil Ekdahl

Lund University, Campus Helsingborg
P.O. Box 882
SE-251 08 Helsingborg
bertil.ekdahl@cs.lth.se

Abstract. The prevailing view of software development as the production of program systems is abandoned in favor of software development as theory building. This suggests that software development properly should be regarded as a deductive activity following the same methodology as is used in all deductive sciences. Programming is merely the description process in the activity of building theories about things in reality. Being the vehicle of theory building, programming may further be considered from a linguistic view. Programs are written in a language and have a proposed meaning; semantics. The main idea is that description and interpretation are complementary in a language; they cannot be fragmented within a language. Seeing programs as linguistic objects in this way has a profound influence on most aspects of software development.

1 Introduction

In [3, p.86] Floyd expresses doubt about "the validity of the established model of thought in software engineering as the sole foundation for our work as computer scientists". Her doubt is, among other things, due to the basic assumption of "software development as the *production* of program systems on the basis of fixed requirements" [ibid]. Floyd thinks that a richer view is needed and the soundest available conceptual model in her opinion is Peter Naur's view of programming as theory building. However, as Floyd points out, "Naur says little about what theory building consists in. And he does not account for the interpersonal nature of communal theory building" [3, p.87].

Naur's paper takes up an interesting aspect of programming which is still not known to many programmers and is certainly not the view that is propagated in software education. Software engineering is still in its infancy and is mainly directed towards methods for program production and little effort is spent on foundational questions. I share Naur's "conviction that it is important to have an appropriate understanding of what programming is" [9, p.253].

In this paper I will give answers to Floyd's questions, both what theory building consists in and the nature of it. Particularly I will spell out the implications of the idea of programming as theory building.

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 100–111, 2003.

2 Naur's View of Programming as Theory Building

In his paper [9] Naur states that he has come to the conclusion of programming as theory building from his observations of "what actually happens to programs and the teams of programmers dealing with them, particularly in situations arising from unexpected and perhaps erroneous program executions or reactions, and on the occasion of modifications of programs" [p.253]. It is not a consideration based on theoretical issues that has led Naur to his conviction but a conclusion that is reached from purely empirical evidence.

Naur is concerned with the "activity of matching some significant part and aspect of an activity in the real world to the formal symbol manipulation that can be done by a program running on a computer" [ibid]. As a support for his view, Naur refers to two examples of which he has experience, examples which are intended to reflect the fact that the program text and its documentation cannot alone explain the underlying ideas. As I will show, this is an observation that in fact is related to the underlying language in which this process of programming is going on. However, Naur does not connect the observation to language. The conclusion drawn by Naur is instead "that at least with certain kinds of large programs, the continued adaptation, modification, and correction of errors in them, is essentially dependent on a certain kind of knowledge possessed by a group of programmers who are closely and continuously connected with them" [9, p.254]. As will be explained, also this observation is a consequence of special properties of language.

Naur suggests that the knowledge built up by the programmer should properly be regarded as a theory in the sense of Ryle's [10]. In this sense "theory is understood as the knowledge a person must have in order not only to do certain things intelligently but also to explain them, to answer queries about them, to argue about them, and so forth" [9, p.255]. The notion of theory employed by Naur is most comprehensive because it also includes "an understanding of the manner in which the central laws apply to certain aspects of the reality, so as to be able to recognize and apply the theory to other similar aspects" [ibid]. Naur uses Newtonian mechanics as an example. A person having this theory "must thus understand how it applies to the motions of pendula and the planets, and must be able to recognize similar phenomena in the world, so as to be able to employ the mathematically expressed rules of the theory properly" [ibid]. Naur argues that in terms of Ryle's notion of theory, "what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program" [ibid]. Naur here touches upon the difference between semantics (certain affairs of the world) and description (theory) but neither he nor Ryle makes a clear distinction between the two concepts. This distinction, that turns the perspective of computers from a mechanical (computational) view to a linguistic view, is decisive for a broader understanding of the idea of programs as theories. This is the perspective I will adopt in the following.

3 What Is a Theory?

As Floyd points out, Naur's view of theory is unclear. In order to unfold the concept of theory I will start to give an explanation of the concept underlying theory, viz. formal system. The connection between formal systems and computers has been well known since the 1930s.

According to Shoenfield [11], a formal system consists of three parts:

1. a language,
2. axioms (formulas in the language),
3. rules of inference.

The purpose of a formal system is to explicate the notion of definability by induction [13, p.2]. This implies that the generated set is at least recursively enumerable.

An inductive definition is a complete mechanical way of generating the elements in a set in the sense that when we know the basic elements in the set we can mechanically generate new members and from these we can, by repeating the same method, generate further members and so on.

There is a correspondence between computability and formal system that was observed by Kurt Gödel already in the 1930s and which he communicated in a letter to Wang [17, p.204] in the following way:

> Formal systems coincide with many-valued Turing machines. The one who works the Turing machine can set a level at each time by his choice. This is exactly what one does in applying a formal system.

Later on Gödel wrote up his observation clearer to Wang [ibid]:

> [A] formal system is nothing but a mechanical procedure for producing theorems. The concept of formal system requires that reasoning be completely replaced by "mechanical operations" on formulas in just the sense made clear by Turing's machines. [...] Single-valued Turing machines yield an exactly equivalent concept of formal system.

From the characterization of formal system follows that the mechanical idea behind it is basic and the equivalence between it and computers (Turing machines) is intelligibly explained by Smullyan [13, p.1]:

> The notions of formal system and mechanical operation are intimately connected; either can be defined in terms of the other. If we should first define a mechanical operation (e.g., in terms of Turing machines), we would then define a "formal" system as one whose set of theorems could be generated by such a machine (that is to say, the machine grinds out all the theorems, one after another, but never grinds out a non-theorem). Alternatively (following the lines of Post), we can first define a formal system directly and define an operation to be "mechanical" or "recursive" if it is computable in some formal system.

A *theory* is a formal system that includes at least first-order logic [5]. Theories are the formal systems that are usually used in mathematics and all other deductive sciences. A particular (first-order) theory is then determined by specifying a set of formulas to serve as its non-logical axioms. For example, arithmetic is a first-order system (theory) with special axioms for the successor function and the functions plus and times. Kleene [6, p.57] has characterized the formal aspect of a theory in the following way:

> A theory in classical mathematics can be regarded as a simple and elegant systematizing scheme, by which a variety of (presumably) true real statements, previously appearing as heterogeneous and unrelated, and often previously unknown, are comprised as consequences of the ideal theorems in the theory.

This idea of *formalizing* a theory is due to the famous German mathematician David Hilbert and involves the total abstraction from the meaning, the result being called a *formal system*. The purpose of a formal system is to make the reasoning completely independent of the underlying meaning. The idea is that derivations of theorems should be completely *mechanical* and not dependent on any meaning of the symbols that become part of the language in which the reasoning is made.

In a lecture, referred to in the book "Conversation with a Mathematician", Gregory Chaitin asks the question what a theory is and gives the following answer: "It's a computer program for predicting observations." [1, p.30].

All computations are theorems in some theory. Since all computations halt it is a basic fact that all computations are computations of recursive functions (computable functions). All such computations can be described, for example, in a system usually called $\mathcal{R}$, the arithmetic system of Robinson[1], which is a much weaker system than the more common Peano arithmetic. If we have a computable function f we may use $\mathcal{R}$ to compute $f(n)$ for every natural number n. Now, suppose that f is a function in physics that describes a ballistic curve, then computing f will predict what will happen for certain input data. The computation is now considered a theorem in physics, albeit still a theorem in arithmetic.

Theories in deductive sciences, possibly with mathematics as an exception, rely on hypotheses that have to agree with observations and should not contradict already known phenomena. The whole purpose of a theory is to, from a small set of premises, be able to make predictions about occurrences that have not yet happened.

4 Theory Building

In all deductive sciences, the way of constructing theories is the same [15]. Since a program is a theory in the deductive sense, programming is no exception. The

[1] See [12, p.338].

most prominent representatives of deductive sciences are without doubt physics and mathematics. They have become the prototype for theory building in other sciences and I will use them as examples to show that theory building in computer science in general and software development in particular does not differ from that of physics and mathematics. It can be argued that mathematics differs from physics in the sense that mathematics does not describe realities. Against this can be asserted that mathematics often takes its impressions from reality but also that mathematics has continued to develop in a "physical" direction once it had found a physical application. It is a remarkable fact that mathematics fits natural sciences in an almost unlikely manner [18]. Furthermore, Platonism is the idea that mathematical concepts are as real as physical entities and as such are waiting for discovery.

Since computer science relies on the mathematical background of recursion theory, it is argued that computer science is applied mathematics. This is of course true but of interest only when we regard the computation as primary. Differently regarded it is applied logic in the sense that programming languages are first order languages that are used to create first order theories about things in reality; the mathematics is secondary. Compare for example with Newton's mechanics. Here mathematics is used only as a means to describe how Newton comprehended the physical world. It is true that without mathematical tools, Newton would have come off badly but yet his intellectual achievement was to give a description of the universe that fitted with observations and could predict physical phenomena in advance.

In physics, theory building usually starts with observations that cannot be satisfactorily explained by old ideas. For example Einstein's observations about general relativity did not fit with Euclid's geometry. In Einstein's case the observations were not based on real experimenting but on theoretical reasoning. Observations put constraints to the physical world that lead to certain hypotheses that give better explanations and exclude other possibilities, at least for the moment. Physicists accept, sooner or later, whatever makes the most sense since nobody knows the blueprint of the universe.

Now, programming also concerns reality. In this case we may say that we know the blueprint even if the opinions may differ among those involved. In order to agree on the reality, requirements play the same role in software engineering as observations do in physics. Requirements put constraints on reality and define the part of reality that is of concern. Like observations in physics, the requirements and their implications are the objects of description. Thus, observations and requirements specification serve the same purpose, viz. to make clear the putative model. The difference lies mainly in the way the model is decided but, as we will see, even here are great similarities.

While software development has much in common with physics, it lacks the experimental part. In physics we have to create the model by observations and guesses and confirm it by experiments while in software development we create the model by requirements and agreements. No experiments are needed in order to establish the future model, it is more a question of delimitation. In software

development we have the ability to completely observe the reality; nothing is hidden and based on hypotheses.

To know the model to be is in both physics and software development a learning process [7]: in physics we get to know the reality by observations, in programming we get to know the reality by requirements. The second difference is that in physics we may start with a description (set of equations) with aesthetic properties, for example elegant symmetry, and then ask for an interpretation (model). By way of example, in string theory in physics it is to a large extent the beauty of the theory that has driven it forth. The advocates of string theory[2] believe that it has revealed a hidden order. Yet, there is still no reality that fits as a model; no experiments have yet been able to give any confirmation from reality.

In software development there is no hidden order in the sense of physics that can be revealed by mathematics but an evolving complexity that is not revealed by mathematical equations but by complete specification of requirements. While the understanding of reality in physics relies on observations and guesses (hypotheses) the understanding of what will serve as a model for the intended theory in software devolvement is more of a social activity. These differences are practical and not fundamental. The way of building theories is in both cases the same.

To illustrate how theory building proceeds I will take the theory of arithmetic as an example. Before being able to characterize arithmetic, that is, to develop the theory of arithmetic, we have to understand the mathematical concepts that will become our model. At this stage we are interested in the structure of (natural) numbers, that is, the mathematical concepts that are part of our intuitive picture of them. For example, our intuition says that there is a first number but not a last number, that we can imagine the numbers as beads on a string where we can come to every bead from the first element by iteration of a generative process. This characterization is a (mathematical) structure[3]

$$A = \langle \omega, <, +, \cdot, S, 0 \rangle$$

such that

1. ω is the natural numbers which are countable,
2. $<$ is a well-ordering on ω with first element 0, successor given by S and such that every element different from 0 has a predecessor,
3. $+$ and $\cdot$ are the ordinary operations *sum* and *product*.

In order to make this structure a model, we have to postulate the properties of the natural numbers, that is, describe how they behave. For example the following axioms (postulates) are taken for granted from our intuitive picture of the numbers.

[2] Or today M-theory which is a development of string theory.
[3] It could be defined more broadly without reference to the natural numbers.

A1. $S(x) = S(y) \rightarrow x = y$
A2. $0 \neq S(x)$
A3. $x \neq 0 \rightarrow \exists y(x = S(y))$

These axioms say that the successor induces an isomorphism between ω and $\omega - \{0\}$. They prevent unwanted properties of the numbers like one with cycles, or with two infinite sequences of elements like

S1. $a_0\ a_1\ a_2\ \ldots b_0\ b_1 b_2\ \ldots$

which is prevented by A3.
However, they leave space for structures like

S2. $a_0\ a_1\ a_2\ \ldots b_{-2}\ b_{-1}\ b_0\ b_1\ b_2\ \ldots$

(which is not prohibited by A3).

So, in order to properly characterize what we have in mind with our structure we have to thoroughly think over how the concepts in it are to behave. If we compare the axiomatization with programming, we can explain the error that S2 is not prohibited as a "bug" in our description (program) of the numbers. If we do not discover it, it may cause unexpected problems. Patching it when discovered after the delivery to the customer may also cause problems. Again we may compare with physics. Patching the old theory of Newton with the idea of relativity created a lot of problems and ended with a complete "rewriting" by Einstein.

When the description satisfactorily describes our intuition of the numbers, the structure A becomes a (standard) model of arithmetic. As is clear, theory building is certainly a description process but the process starts with making a clear structure, that is, a clear conceptual idea of the reality which will become the objective of the description. If the reality is not clear or not well understood it is easy to envisage that the program (description) has no distinguishing model, or differently expressed, what the program expresses is not true in the reality that the program is intended to map; its semantics is incorrect. Tarski [14, p.401] expresses the necessity of having a clear conception of reality in the following way:

> We shall understand by semantics the totality of considerations concerning those concepts which, roughly speaking, express certain connexions between the expressions of a language and the objects and states of affairs referred to by these expressions. [...] The concept of *truth* also – and this is not commonly recognized – is to be included here, at least in its classical interpretation, according to which 'true' signifies the same as 'corresponding with reality'.

To summarize: theory building starts with a conceptualization of reality, giving a *structure*, and ends with a description, the theory (program). If all computations in the program are true in the structure, the structure is a model of that program.

5 Complementarity in Language

Naur makes the observation that "the program text and its documentation has proved insufficient as a carrier of some of the most important design ideas" [9, p.254]. This is supposed to be due to the fact that programming involves building up the programmer's knowledge which is argued to go beyond what can be described in programs and text documents. The dependency of a theory is stated by Naur in the following way [9, p.255]:

> The dependence of a theory on a grasp of certain kinds of similarity between situations and events of the real world gives the reason why the knowledge held by someone who has the theory could not, in principle, be expressed in terms of rules.

Naur does not further explain why just similarity between situations and events of the real world is inexplicable; he just states it. However, if we observe that Naur in fact talks about incompleteness concerning language, then the tension between what can be described and what remains as the programmer's knowledge can be explained as a consequence of a holistic view of language which Löfgren [8] calls *the linguistic complementarity*. It is the thesis that the semantics of a language cannot properly be described in the language itself. It is a whole of complementary description and interpretation processes; both are needed to form a language. This thesis of Löfgren is supported by results concerning formal languages. For example, Tarski [15] has shown that truth in formal systems cannot be expressed in the language itself, meaning that the semantics of a program is not part of the program. The semantics is in the mind of the programmer and can only be expressed in a metalanguage which must possess a higher order than the programming language and according to Tarski [15] cannot be expressed in the programming language.

What Löfgren and Tarski refer to is that between description and semantics there is a process, an interpretation process, that transforms descriptions to meanings and that this process is not describable in the language itself. This interpretation process must not be taken as a function that transforms words, or sentences, in the description to a definite meaning. For a formal language it has a definite meaning in a specific model but there are several "realities" (structures) that each can serve as a model of a given description.

In the complementarity view of language, a language consists of four parts: *description* (theory), *interpretation* (semantics), *interpretation process* and *description process*. The interpretation process (I_P) and description process (D_P) act as is illustrated in Fig. 1. The linguistic complementarity can now be stated as referring to holistic situations where fragmentation into parts does not succeed. Language is a whole of description and interpretation, parts that are not expressible within the language itself.

Now, the question becomes more intricate when we ask why not all people, having the same language, have the same interpretation. Swedish, for example is a language and all Swedes have Swedish as their native language. But even

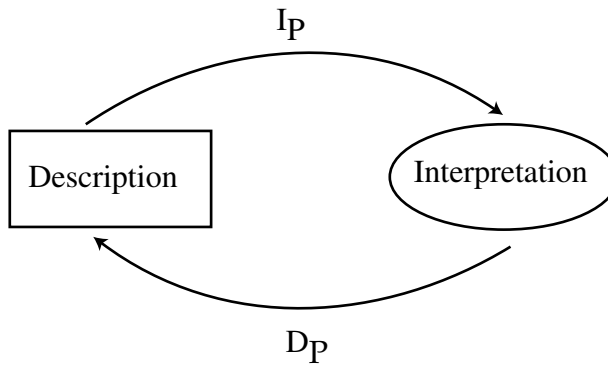


Fig. 1. Components in a language

within a language, one and the same description can be differently interpreted. If we stick to the holistic view above of language we may express it as everyone having his or her "own" (Swedish) language where most features are in common. Thus, the interpretation is a process that belongs to the individual's language, that is, to the interpreter. It is true that in order to be social beings we have to have almost the same interpretation of the language. The differences are most noticeable in the outskirts of the common social area. Misunderstandings are due to different interpretation processes. However, a characteristic trait of the human brain is that it is easy to adapt one's interpretation to another individual's and also merge the processes from several individual's to a common interpretation. This is, for example, the aim of political speeches[4].

The understanding of complementarity in language gives a quite different understanding of programming but also gives a different explanation of many of the problems involved in software development. The main reason for faulty programs and programs that do not fulfill the customers' requirements and in many cases are so defective that the whole project will never be completed is the lack of a common model, that is, a well specified model[5] (structure) that can be shared by everyone. About 80-90% of the software development does not meet its performance goal and as much as around 40% of the development fails or is abandoned[6].

In the development stage it is the model (structure) that should be in focus and should be given such a description that everybody understands it. If not everyone in the project has the same conceptualization of the reality (structure) it is impossible to produce a (formal) description that unambiguously points to a certain structure, that is, has a distinct model; a standard model. If many people

[4] The social aspects of language are treated in detail in [2] and [4].

[5] Strictly speaking it is the structure, not the model, which is of concern. Since we are talking about the structure that will become the model of the future program, I do not always distinguish between the two entities.

[6] OASIG, 1996.

in a project have different opinions of reality, those different structures cannot be models of the final programs. Not even the programmer has always a clear conception of the program and accordingly has no model that is in agreement with reality.

Despite the striking resemblance to theory building in physics, there is in a certain respect a decisive difference. All physicists share the prevalent models of nature, except in a paradigm shift. The physicists are from their early education trained to understand the different models used in physics. Those who cannot understand how nature is supposed to be, are sorted out at an early stage. The models are understood by thoroughly working out the ideas until one can grasp it as a whole and see it as a single idea. This process is not abandoned until it has yielded a full comprehension of the ideas. So, there is no problem when physicists from different places or countries come together. They already share the models. This is in sharp contrast to programmers. Here the analysts normally have different background knowledge compared with the customers and even between themselves. Despite that, they are supposed to grasp an idea of an information flow that is in agreement with the customer's reality. Notice, it is not enough to have a common theoretical knowledge, for example of logistics. This does not give a sufficient understanding of its application in a certain industry.

As already mentioned, from his observations Naur draws the conclusion "that at least with certain kinds of large programs, the continued adaptation, modification, and correction of errors in them, is essentially dependent on a certain kind of knowledge possessed by a group of programmers who are closely and continuously connected with them" [9, p.254]. What Naur describes here is a maintenance problem which easily can be given an explanation in terms of the linguistic complementarity. When a new programmer enters an existing project he or she has not the same background knowledge as those that already are in the project or those who have left it. He or she has not the common model but has to learn it and this cannot be done from descriptions only. It takes time to comprehend the model and there is no short cut; it is a cognitive process that has to mature individually.

With the help of the linguistic complementarity, we can identify important reasons for faultiness. First, the programmer does not sufficiently well appreciate the model, second, if the model is changed this is not always properly communicated to the project members. In both cases the result is that the project members do not have the same model but believe they have. This may end in different theories possibly correct in their respective models but inconsistent between themselves.

6 Conclusion

In formal logic and computer science, the term language has a very narrow meaning. A language is considered completely specified when its symbols and formulas are specified. This makes a language a purely syntactic object [11, p.4]. However, a program, as well as a formal system, is constructed in order to

have a meaning. The whole point with a language is to be able to communicate ideas or facts about a real or conceptual world. In computer science, program language is normally not thought of in those terms. It is merely a tool to produce computations. The semantics is put in the background.

If we change the view of programming to have theories as its outcome, it will influence our conception of programs. This view is a shift from a production perspective to a linguistic perspective in which the communicative aspect of language is put forth, that is, the semantics will be a central part in programming. Syntax and meaning together completely specify a language making it a holistic entity that cannot be fragmented into parts in the language itself; description and interpretation are complementary within the language.

This holistic view of language, called the linguistic complementarity, will change our view of programming in a profound way since it will change the focus from programs (description) to semantics (model). The whole point with a program (theory) is to formalize something (reality) so precisely that arguments about it can be followed mechanically. Only in that way can we be really sure of their validity.

The idea of programming as theory building will also influence the methods used in the development and maintenance of programs as well as our view of software architecture. The big challenge is to find good methods to describe the structure that will become the model of the program.

References

1. Chaitin, G. J. 2002. *Conversations with a Mathematician, Math: Art, Science and The Limits of Reasoning*, Springer-Verlag, London Ltd..
2. Ekdahl, B. 2001. Can Computers be Social? *The Fifth International Conference on Computing Anticipatory Systems*, (HEC-Lige, Belgium, Aug).
3. Floyd, C. 1992. Software Development as Reality Construction. *Software Development and Reality Construction* (Floyd C., Züllighoven H., Budde R., Keil-Slawik, R., Eds.), Springer-Verlag, Berlin.
4. Gärdenfors, P. 1993. The emergence of meaning. *Linguistics and Philosophy 16*, 285-309.
5. Haskell, B. C. 1963. *Foundations of Mathematical Logic*, McGraw-Hill Book Company, Inc., Dover edition 1977, Dover Publications, Inc. NY.
6. Kleene, S. C. 1952. *Introduction to Metamathematics*, Tenth impression 1991, Wolters-Noordhoff Publishing-Groningen, North-Holland Publishing Company, Amsterdam.
7. Löfgren, L. 1982. Methodologies and the induction problem. *Progress in Cybernetics and Systems Research*, vol. VIII (Trappl R., Klir G., Pichler F., Eds.), Hemisphere, London, 15-22.
8. Löfgren, L. 1991. The nondetachability of language and linguistic realism. *Constructive Realism in Discussion* (van Dijkum, C., Wallner, F., Eds.), Amsterdam, Sokrates Science Publisher.
9. Naur, P. 1985. Program as theory building. *Microprocessing and Microprogramming 15*, North Holland Pub. Co., 253-261.
10. Ryle, G. 1963. *The Concept of Mind*, Penguin Books Ltd., Harmondsworth, England.

11. Schoenfield, J. R. 1967. *Mathematical Logic*, Addison-Wesley Publishing Company.
12. Smorynski, C. 1991. *Logical Number Theory I*, Springer-Verlag.
13. Smullyan,, R. M. 1961. Theory of formal systems. *Annals of Mathematical Studies 47*, Princeton Univ. Press.
14. Tarski, A. 1983. The establishment of scientific semantics. In [15].
15. Tarski, A. *Logic, Semantics, Metamathematics, papers from 1923 to 1938* (2nd ed.) (Tarski, A., Translated by Woodger, J. H.), Hackett Publishing Company, Indianapolis.
16. Tarski, A., 1994. *Introduction to Logic and to the Methodology of the Deductive Sciences* (4th ed.) (Tarski, J., Ed.), Oxford University Press.
17. Wang, H. 1996. *A Logical Journey: From Gödel to Philosophy*, The MIT Press.
18. Wigner, E. P. 1960. The unreasonable effectiveness of mathematics in natural sciences. *Communications in Pure and Applied Mathematics 13*, 1-14.

In Search of a User Base: Where Are the B's?

David L. Hicks

Department of Computer Science
Aalborg University Esbjerg
Niels Bohrs Vej 8
6700 Esbjerg, Denmark
hicks@cs.aue.auc.dk

Abstract. Research in the open hypermedia systems area has been ongoing for the past several years. In spite of the research progress that has been made in the field, however, open hypermedia systems in general have yet to develop a large and substantial user base. This situation is particularly conspicuous when considering the increasing popularity of similar "B-level" technologies such as object management frameworks. The range of factors contributing to the missing user base of open hypermedia systems includes potential problems with standards, publicity, and the WWW. In spite of these limiting factors, however, new research trends in the open hypermedia systems area such as structural computing offer capabilities that might be exploited in order to more clearly and effectively communicate the advantages of the open hypermedia approach to potential users of the technology.

1 Introduction

For the past several years, researchers have been actively investigating open hypermedia systems [9,11,12]. The original motivation for research into the open hypermedia area was to address the problems that were identified with early hypermedia architectures. In particular, researchers focused on strategies to provide hypermedia functionality in a way that would enable it to be used by a wide variety (or open) set of applications. A primary goal was to enable existing applications to readily incorporate basic hypermedia functionality in order to augment the services they provide [7]. Much progress has been made in the open hypermedia research area. A number of open hypermedia systems have been successfully developed. Examples include DHM [4], Microcosm [5], Chimera [1], and many others. In addition to basic hypermedia functionality, many open hypermedia systems provide the infrastructure to support additional important capabilities such as distribution and collaboration. Progress has even been made on creating standards for open hypermedia systems to allow them to interoperate [2].

In spite of the research success of open hypermedia systems, in general they have yet to find a large and substantial user base. It is true that almost all open hypermedia systems have client applications that have been developed or integrated to test their functionality. These applications demonstrate the usefulness

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 112–117, 2003.

of the services the systems provide. However, the number of client applications for any particular open hypermedia system is usually small, and is often limited to the set of "proof of concept" applications needed for research purposes. Though commercial versions of some open hypermedia systems have been developed, application developers have yet to line up in large numbers, eagerly awaiting the opportunity to integrate the services that open hypermedia systems provide.

This paper examines the limited user base of open hypermedia systems. The following section provides a brief look at the different types or levels of work in order to characterize open hypermedia system users. Section 3 provides an initial look at some of the factors that may have limited the use of open hypermedia systems. Section 4 examines one of the most recent trends in open hypermedia systems research, structural computing, and the role it might play in helping to enlarge the user base of this technology. Section 5 concludes the paper.

2 Levels of Work

Work in computer science (as well as other areas) is sometimes described as occurring at different levels, according to the primary beneficiary of the efforts represented by the work. For example, Douglas Engelbart has suggested that the work which occurs within organizations can be classified according to a three level classification scheme [3]. Work occurring at the first level (called the "A-level") represents those efforts that are directly related to the primary function of a company or organization. For example, in an automobile design firm, A-level work would correspond to the efforts made by designer personnel to produce new car designs. The second category of work, called "B-level" work, represents the efforts of those people whose job it is to build tools for and support the A-level workers. In the automobile design firm example, B-level workers would include those who build CAD packages and other tools that directly support the designers who design cars. The third category of work, "C-level" work, corresponds to the people who build tools to facilitate B-level work. This category would include those who build infrastructure frameworks for object management, collaboration, distribution, etc., tools that can facilitate the development of B-level tools such as a CAD package.

When considered within the context of the three level work classification scheme, much of the research in the open hypermedia systems area can be considered to be C-level work. Consider the functionality provided by open hypermedia systems. The services they provide are not intended to be directly used by end users (A-level work). Indeed, by design, services in an OHS environment usually have no specific user interface. Instead, the services provided are intended to be incorporated into applications by application developers in order to augment the functionality they provide to end users. In this way, open hypermedia systems are tools designed to be used by B-level workers. That is, they are "tools for the people who build tools for other people".

The observation concerning the missing user base for open hypermedia systems and structural computing environments identified in the previous section can now be expressed as "where are the B's"? Where are the users who develop applications and other tools for A-level work that could potentially benefit from the services provided by open hypermedia systems? This question is the topic of the next section.

3 Where Are the B's?

Some C-level tools have been quite successful and managed to accumulate a substantial B-level user base. For example, consider object management frameworks such as Java RMI or those based on the CORBA standard. They are often used in application development, and their popularity and usage continue to increase. Indeed utilization of a specific object management framework is sometimes even used as a selling point for applications. For example, vendors sometimes advertise their products as being compliant with some specific standard.

The success of some C-level technologies makes the absent user base for open hypermedia systems even more conspicuous. What could it be that has hindered the wider spread usage and adoption of the open hypermedia approach? Could it be some fundamental flaw with the open hypermedia way of providing hypermedia services? Possibly, but the open hypermedia systems approach to providing hypermedia functionality is not an immature or untested idea. It is based upon over a decade of research into hypermedia architectures. These research results have been reviewed by and acknowledged within the general hypermedia research community. For the past several years, open hypermedia research results have been well documented in the literature. Though it is difficult to know exactly what the problem might be, from a research perspective it is appealing to believe it does not lie in the basic open hypermedia approach. Is there instead some set of other (external) factors that have prevented the development of a larger user base for open hypermedia? The remainder of this section will examine an initial set of factors that could be contributors.

Standards. Could it be a lack of standards that has discouraged potential application developers from integrating open hypermedia functionality? It is easy to see how this would be an important question for developers, causing them to stop and consider whether integrating open hypermedia services was worthwhile. Would the system or set of systems they plan to integrate with provide the ability to interact with other systems? Of course, there have been standardization efforts within the open hypermedia community, most notably the Open Hypermedia Protocol (OHP) developed by the Open Hypermedia Systems Working Group (OHSWG) [2]. Even though demonstrations have successfully shown open hypermedia system interoperation based on the OHP, is it still somehow not enough? Was it not sufficiently formalized or sanctioned by the appropriate standards bodies to generate confidence within the minds of application developers?

Publicity. Certainly open hypermedia research results have been well publicized in the research world. But has this publicity somehow failed to reach and influence the application developer community? Most open hypermedia research efforts have been conducted at the basic research level, and the results are most often published in corresponding venues. It would be reasonable to expect a certain time lag before the research begins to influence application developers. However, considering the number and maturity of some open hypermedia system projects, it would seem reasonable to expect more of an influence to have occurred by now.

The WWW. Are there web influences at play that have served to stifle the impact of open hypermedia research? Despite its limitations, the web is certainly the mostly widely used form of hypermedia, and can have a significant influence. Before the advent of the web, for most people the word hypermedia was a curious research term. Now almost everyone has a notion of what hypermedia is. Unfortunately, many people today equate the term with the navigational hypermedia capabilities of the web. Has the term been co-opted and redefined to the point that people can no longer see past the limited capabilities of the web to the advantages of more advanced forms of the technology such as open hypermedia systems?

4 Trends in Open Hypermedia Research

One of the latest trends in the open hypermedia systems research area is structural computing. Structural computing seeks to expand the types of services offered within open hypermedia environments to include more than just the traditional or "navigational" form of hypermedia. In the past few years, researchers have identified new types of hypermedia that have important roles in supporting users. These new forms of hypermedia move beyond the traditional node-link model. Examples include spatial hypermedia [6] and taxonomic hypermedia [10]. The goal of structural computing systems is to provide support for several types (or domains) of hypermedia within the same environment. The intent is to provide a rich platform for application developers, enabling them to choose from among many different types of hypermedia when augmenting their applications, and to even incorporate functionality from multiple hypermedia domains within the same application when appropriate. Though a relatively new branch of open hypermedia systems research, the first structural computing environments have already begun to emerge. Examples include Callimachus [13], FOHM [8], and Construct [14].

As one of the newest trends in open hypermedia research, can structural computing play a role in helping expand the user base of open hypermedia technology? The emphasis in structural computing on multiple hypermedia domains would seem to help. Firstly, an environment offering support for multiple hypermedia domains is functionally richer and a more attractive one in which to integrate applications than one supporting only a single domain. Secondly, and importantly, the support for multiple domains can help address one of the most

significant limiting factors identified in the previous section, the influence of the web. Support for multiple domains is clearly something different from the navigational hypermedia capabilities of the web. This distinction might make it easier for researchers to communicate to B-level developers the advantages that the open hypermedia/structural computing approach has to offer beyond what the web can provide.

5 Conclusion

Despite significant research attention, open hypermedia technology has yet to develop a large-scale user base. Considering the success that some other C-level technologies have experienced, it is important to examine what the limiting influences on open hypermedia systems might be. Identification of limiting factors will enable the most effective exploitation of new open hypermedia research directions, such as structural computing, in the search for a B-level user base for open hypermedia technology.

References

1. Anderson, K. M., Taylor, R., and Whitehead, E. J. 1994. Chimera: Hypertext for heterogeneous software environments. *Proceedings of the 1994 ACM European Conference on Hypertext*, (Edinburgh, Scotland, Sep), ACM Press, 94-107.
2. Davis, H. C., Millard, D. E., Reich, S., Bouvin, N. O., Grnbk, K., Nürnberg, P. J., Sloth, L., Wiil, U. K., and Anderson, K. M. 1999. Interoperability between hypermedia systems: The standardisation work of the OHSWG. *Proceedings of the 1999 ACM Conference on Hypertext*, (Darmstadt, Germany, Feb), ACM Press, 201-202.
3. Engelbart, D. C. Hypertext'98 Open Hypermedia Systems Workshop Keynote Address.
4. Grønbæk, K. and Trigg, R. H. 1994. Design issues for a Dexter-based hypermedia system. *Communications of the ACM 37*(2) (Feb), 40-49.
5. Hall, W., Davis, H., and Hutchings, G. 1996. *Rethinking Hypermedia – The Microcosm Approach*, Kluwer Academic Publishers.
6. Marshall, C. and Shipman, F. 1995. Spatial hypertext: Designing for change. *Communications of the ACM 38*(8), 88-97.
7. Meyrowitz, N. 1989. The missing link: Why we're all doing hypertext wrong. *The Society of Text: Hypertext, Hypermedia and the Social Construction of Information*, MIT Press, 107-114.
8. Millard, D., Moreau, L., Davis, H., and Reich, S. 2000. FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains. *Proceedings of the 2000 ACM Hypertext Conference* (San Antonio, TX, Jun), ACM Press, 266-267.
9. Nürnberg, P. J. (Ed.). 1999. *Proceedings of the First Workshop on Structural Computing*, Technical Report CS-99-04, Department of Computer Science, Aalborg University Esbjerg, Denmark.
10. Parunak, H. 1991. Don't link me in: Set based hypermedia for taxonomic reasoning. *Proceedings of the 1991 ACM Hypertext Conference* (San Antonio, TX, Dec), ACM Press, 233-242.

11. Reich, S. and Anderson. K. M. (Eds.). 2000. *Proceedings of the Second International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 1903), Springer Verlag.
12. Reich, S. Tzagarakis, M., and De Bra, P. (Eds.). 2001. *Proceedings of the Third International Workshop on Structural Computing*, Lecture Notes in Computer Science (LNCS 2266), Springer Verlag.
13. Vaitis, M., Papadopoulos, A., Tzagarakis, M., and Christodoulakis, D. 2000. Towards structure specification for open hypermedia systems. In [11].
14. Wiil, U. K. and Hicks, D. L. 2001. Tools and services for knowledge discovery, management and structuring in digital libraries *Proceedings of the Eighth ISPE International Conference on Concurrent Engineering: Research and Applications*, (Anaheim, CA, Jul).

Reconciling Versioning and Context in Hypermedia Structure Servers

Jon Griffiths, David E. Millard, Hugh Davis, Danius T. Michaelides, and Mark J. Weal

Intelligence, Agents, Multimedia Group
Dept. of Electronics and Computer Science
University of Southampton, U. K.
{jpg96r, dem, hcd, dtm, mjw}@ecs.soton.ac.uk

Abstract. Contextual structure servers and versioning servers share a similar goal in allowing different views on a stored structure according to the viewer's perspective. In this paper we argue that a generic contextual model can be used to facilitate versioning. In order to prove our hypothesis we have drawn on our experiences with OHP-Version to extend FOHM's contextual model.

1 Introduction

There is a long standing tradition of versioning structure within hypermedia systems [29,30,35]. Versioning provides a failsafe baseline for experimentation. Hypertext authors can rollback the hypertext network to its previous state after changes have been made to it. It is also useful for supporting hypermedia collaborative work when needing to freeze versions where many users are working on a shared resource. Versioning is particularly useful in application areas where inter-document relationships cannot afford to be lost, such as in legal and audit documents.

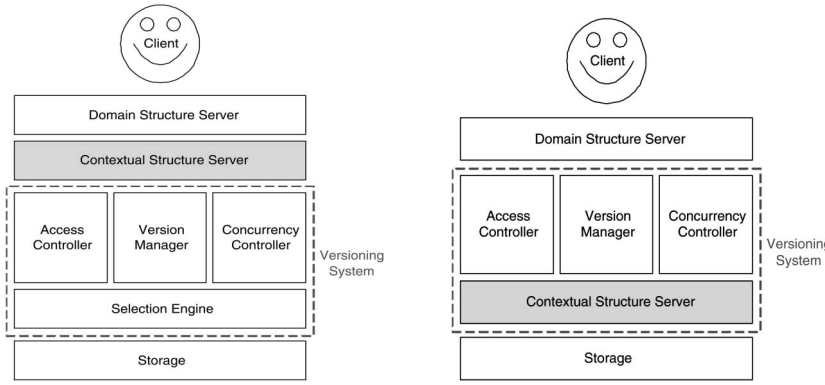
More recently there has been work focusing on contextual structure servers [17,2]. These dynamically generate different views of complex structure (such as a hypermedia linkbase) according to the context of the viewer.

A contextual server allows for many different views on a given structure, depending on the viewer's position in n dimensional context space (examples of contextual dimensions include: age, expertise, goals, interaction history, etc.) Since time can be viewed as just one of these contextual dimensions it seems intuitive that any complete contextual model should be powerful enough to drive the selection mechanism of a versioning server.

This is not just functional re-use. It recognises that versioning is a specific kind of contextualisation and that there is an equivalence between different versions of an object and different contextual views of an object.

Figure 1(a) shows the current view of how contextual structure servers and versioning systems interact. The selection engine of the versioning system is conceptually a separate process. Figure 1(b) shows what the reconciled architecture

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 118–131, 2003.



(a) Current Architecture (b) Reconciled Architecture

Fig. 1. Architectures for placement of contextual structure servers with versioning systems.

would look like when the generic services of a contextual structure server replace those of a specific selection engine.

It is our opinion that reconciling versioning and contextualisation mechanisms would result in a more natural and consistent approach to both problems.

1.1 Objectives

At Southampton we have been experimenting with a contextual structure server known as Auld Linky [16]. It is a simple stand-alone server that stores hypermedia structure expressed in the Fundamental Open Hypermedia Model (FOHM) [19], a generic contextual hypermedia format that evolved from the OHP suite of interoperability protocols.

Separate to this we have also been developing a proposal for OHP-Version, a versioning framework and protocol to control the versioning of OHP-Nav links.

In this paper we use FOHM and Auld Linky to ground a discussion of how a contextual structure server could be used to provide the kind of versioning supported by OHP-Version.

During this process we answer several questions:

1. Can a contextual model be powerful enough to provide versioning support (i.e. can a contextual structure server act as the selection engine behind a versioning service)?
2. We will answer this by looking at whether FOHM's contextual model is powerful enough to support versioning, and if not, how can it be extended so that it is?
3. What are the consequences of such a contextual model for the non-versioning applications of context?

2 Background

Frank Halasz discussed the issues of hypermedia versioning at the Hypertext '87 conference [11]. He highlighted the need to provide version support for both hypermedia documents and structure.

One of the earliest systems to consider versioning was Xanadu [20]. Versioning is used as a device to prevent unnecessary duplication of content between document versions. For example if a document contains the same content as a predecessor version then instead of containing a duplicate copy of that content it will contain content links that point to the same content within the older document.

Notable systems that directly tackle the problem of versioning hypermedia structure include CoVer [8], VerSE [7] and HB3 [12]. CoVer and VerSE implement policies that employ versioning as a tool for aiding users when performing a task (such as writing a research paper). These systems are also known as contextual versioning systems, but they have a very different notion of context compared to that described within this paper. Their meaning of context is a description of the task being performed when the objects (involved in carrying out that task) are versioned. Such context information is stored in order to assist with the version identification process.

The HB3 system, on the other hand, focuses on versioning support at the hyperbase level. It enables versioning of navigational hypermedia structure: nodes, links, composites and whole hypertext networks. HB3 can support a wide variety of versioning paradigms as versioning applications can build on this low-level versioning capability to construct more complex versioning styles.

The success of the World Wide Web has also led to an interest in web-based versioning. But the embedded nature of the Web means that hypermedia structure can only be versioned implicitly by versioning those web documents that contain the embedded structure. The most well known system is WebDAV (Web Distributed, Authoring and Versioning) [31]. It extends HTTP to provide complete version control over Web documents, including the ability to retrieve, delete, copy, move and lock both document revisions and their associated revision properties.

2.1 OHP-Version

The Open Hypermedia Systems Working Group (OHSWG) has spent much effort on addressing the problem of interoperability between Open Hypermedia Systems (OHS). Their work culminated in the creation of OHP-Nav, a standardised linking protocol that allows components of an OHS to discuss navigational hypermedia [4,18,22].

The IAM group at Southampton has been investigating how to supplement OHP-Nav with versioning functionality, leading to the creation of the OHP-Version protocol. This protocol has four goals:

1. To store and select different revisions of OHP hypermedia objects (nodes, anchors, endpoints and link objects).

2. To preserve the state of composite structures (such as the collection of OHP-Nav objects, and connections between them, that together form a link structure).
3. To capture the state of the entire hypertext network at a given moment in time by versioning a collection of hypertext structures.
4. To maintain the version history of OHP hypermedia objects, composite structures and the hypertext network.

The approach taken when developing OHP-Version was to store all versioning data in a separate version server rather than in the OHP-Nav linkserver itself. This was to ensure that minimal disruption was caused to non-versioning clients when introducing versioning to the OHP environment. However some minor changes still had to be made. For example to ensure that OHP-Nav clients can still retrieve the latest revisions of hypermedia objects it was necessary to append a new property, called the 'default status' attribute, to all OHP objects. In addition, the OHP-Nav linkserver had to be re-programmed so that it permits non-versioning clients to manipulate latest revisions only.

3 Alternative Versioning Strategies

Auld Linky [16] is a stand-alone structure server that serves FOHM structures expressed in XML. Queries are sent to Linky in the form of a FOHM XML pattern which is matched against Linky's loaded structures. Those that match are returned.

Linky also supports FOHM's context model. Queries can be sent with an associated context description, in this case Linky matches the pattern, but then culls away any parts of the structure whose context objects do not match that of the query. It then checks that the culled structure still matches the original pattern before returning it.

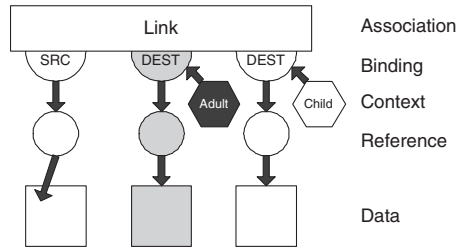


Fig. 2. A Contextual Link in FOHM

Figure 2 shows how parts of a Navigational Link are culled away. In this case the query has requested all link structures given a context with keyword *age* and the value *Child*. The first link destination has a context where age is set to

Adult and thus the context match fails (represented in black) this results in that entire branch of the link (represented in grey) being removed before the link is returned.

Our purpose is to reconcile this kind of contextual model with versioning. In this section we compare several strategies for versioning given the facilities of a contextual structure server.

3.1 Naming Schemes

Perhaps the most pragmatic versioning approach would be to employ a naming scheme to capture the version history of individual hypermedia objects, similar to that of the Concurrent Version System (CVS). The nature of a naming scheme is to make the id of each object a meaningful name that imparts information about the version history of the objects in terms of how it is related to other versions. Thus the format for a naming scheme for FOHM could be: {`object_id:revision_id`}. The object id is a unique name that is shared amongst objects that belong to the same versioned object. It indicates which objects are related to one another. The revision id is a name that is unique amongst the objects. It describes the object's location within the version tree.

Because complex FOHM structures involve so many named objects (for example, there are six named objects in a basic two-sided navigational link), Auld Linky allows linkbase authors to create anonymous objects. These have no first class status of their own and instead are considered as part of their parents. This is a problem because to correctly version all the objects in a linkbase (without massive duplication) they must all be named.

The easiest way to achieve this would be for a structure server to automatically name all anonymous objects at the time of their creation within the server (for example, when a static linkbase was loaded). This would solve the naming problem within the server while still allowing linkbase authors the luxury of anonymous objects.

From the revision id the identity of an object's parent, children and siblings can be deduced. The revision id would also enable the order of the objects, in terms of eldest to youngest, to be calculated. Such a naming scheme could easily be implemented to capture the version history of hypermedia objects. However, this versioning solution does not help to reconcile the role of context in versioning structure.

3.2 Concepts

One strategy that we have used successfully in the past to model different views of an object is to explicitly model the relationship between them using a *concept* object [2]. These are Associations of type 'concept' that relate several objects that represent the same conceptual entity.

Context objects attached to the bindings of the Association implement conditional membership of the concept. Thus the same concept can be seen differently according to the context of the viewer. This could even transform the object into

totally different structures, for example a biographic description might be a short piece of text in one context (a single Data object) but become a larger composite document (made up of a tree of Associations) in another.

The concept structure that we have previously used does not contain enough information to capture the evolution of its members (i.e. it contains no ordering information). This is necessary with versioning as it is important to record the order in which versions were created. In addition, version structures may branch (i.e. any given version of a structure might spawn many new versions). To model this requires a tree structure rather than an ordered set.

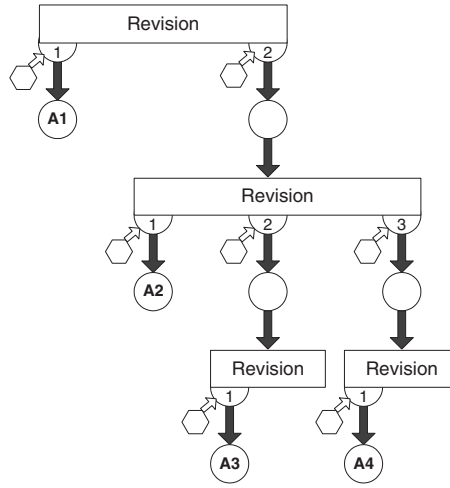


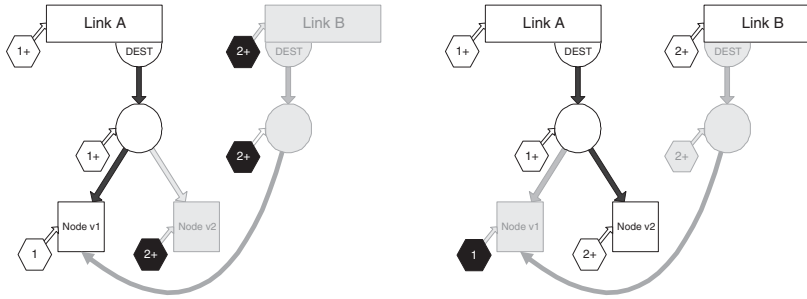
Fig. 3. A Revision Tree in FOHM

Figure 3 shows a tree of *revision* structures that manages to capture the version history of an object (in this case a Navigational Anchor, represented in FOHM by a Reference object). Each revision structure stores an ordered list of revisions, where position 1 represents the original.

Unfortunately a tree structure is cumbersome to manipulate and difficult to query, especially one that is n levels deep. This makes concept structures an awkward strategy for versioning. Ideally the tree would be transparent to a query, which indicates that concept structures should be treated as more fundamental, perhaps as part of the contextual model itself.

3.3 Multiple Connections

Amending the FOHM data model to permit many-to-many relationships between all its objects presents an opportunity for another approach to versioning. In this scenario each FOHM object operates as its own concept structure with context determining which objects are connected at run-time.



(a) Query in Context 1 (b) Query in Context 2

Fig. 4. Contextual Queries with Multiple Connections. The black contexts are the ones that have failed, this results in the grey structure being trimmed away

Figure 4(a) shows two Link structures where the context objects describe in which time slice the object is valid. The number within the context object refers to the time slice that the object was created in, a plus sign denotes that the object is visible in all subsequent time slices.

Link A has been created in time slice 1, a second version of Node v1 has then been created in time slice 2 (Node v2). When the structures are queried in the Context of time slice 1, Node v2 is removed and the viewer sees the older version of the link structure.

Link B has also been authored in time slice 2. The author explicitly wants to refer to the older version of the Node (perhaps they are discussing a particular draft of a document) and so creates a reference to Node v1. When the query is sent in the context of time slice 1, Link B is totally invisible.

Figure 4(b) shows the same structure queried in the context of time slice 2. In this instance the latest version of the Link A structure is returned (including Node v2) and the Link B object is visible. The problem is that since Node v1 is invisible in this context it does not appear as part of the Link B structure, even though this was the author's intention.

The inability to refer to older versions of objects is a fundamental problem with this strategy. The source of the problem is that FOHM's contextual model only allows objects to be contextual, whereas we require the *connections* between objects to be contextual. In the next section we explore what a contextual model that includes contextual connections would look like.

4 Contextual Connections and Pattern Propagation

Having explored the various strategies we have discovered that FOHM's contextual model is not powerful enough. To support versioning a contextual model requires contextual connections as well as objects that describe in which context they are visible. In this section we look at how connections are formed within FOHM and how we might make them contextual.

4.1 Patterns

FOHM Objects may be referenced by a unique id, i.e. pattern match for a given type of object with a particular id. Because the ids are guaranteed to be unique, Linky can circumvent the matching process and use more efficient lookup algorithms (such as a hash table).

If we want this look-up to become contextual we need to allow non-unique ids and full patterns to be used (rather than just a direct id lookup). This would force the structure server to use the same pattern matching process internally as clients of its external services. We have called this approach *pattern propagation*. It allows a pattern to be attached to a static structure to be resolved at run-time during the query process.

We can generalise this internal pattern match further by allowing query contexts to be attached to the stored patterns. This extension allows structures to be connected together *in context*. For example, a link could be authored to a node in the context of a particular time slice. When the link is retrieved it will include the revision of the node from that time slice, even if the original query was made in the context of the current time slice.

4.2 Storing the Patterns

In the contextual structure server that we are using, Auld Linky, the id lookup objects are currently stored with a simple flag to show that they are to be used as an id lookup as opposed to being first class objects themselves.

Replacing this with a pattern to be further matched is not as simple as adding additional elements, as the context for the pattern propagation must also be stored. Furthermore we need to allow queries to be made independently of type (i.e. a reference should be able to store a pattern that resolves into *either* a Data object or an Association).

This would require two extensions to the current Linky FOHM definition. Firstly, it must become possible for authors to create a *ReferencableObject* as well as an Association or Data object. This would match with either of the other two. Secondly all FOHM's first class objects need to be able to contain *QueryRules* when written in their pattern state. This would include the context in which to make the query.

Once these patterns have been stored in the linkbase we are then faced with the problem of getting them out again (since a query would cause them to be automatically resolved). One solution would be to place a flag in the query rules that specified the *depth* to which patterns should be resolved. A depth of n would return all patterns, while adjusting the depth to other numbers would allow an application to retrieve the uppermost parts of large structures without invoking potentially expensive pattern matching processes on the lower parts (for example, to return a tour structure without processing the members of that tour).

4.3 Merging Contexts

When a contextual query is sent to the structure server it already has a Context object attached to it. With the current context model this single context is used throughout the matching process, but with pattern propagation it may change as the matching process works its way down different branches of the structure.

This means that there is an issue of how the query contexts and the pattern contexts should be merged. There are four choices:

1. *Replace the Query Context.* The pattern context could entirely replace the query one.
2. *Merge Contexts (replace Pattern Keys).* The query and the pattern context could be merged; the query context's keys' values taking precedence over the pattern context's keys' values.
3. *Merge Contexts (replace Query Keys).* The two contexts could be merged; the pattern context's keys' values taking precedence over the query context's keys' values.
4. *Merge Contexts (conjoin keys).* The two contexts could be merged with the values of like keys being AND'ed together.

Without merging the original context information would be discarded even if it has no bearing on the new context information. Conjoining contexts may prove to be very difficult, particularly as Linky allows values to be *constrained* (matched via included code) and AND'ing arbitrary code is non-trivial. However, one might imagine situations where both the remaining merging scenarios are required.

For example, when using context to control children's access to adult material it seems sensible to maintain the existing keys (i.e. the age is always set to child, thus a child cannot 'move' into an adult context). However, when going the other way it makes sense to allow the context to be overridden (allowing adults to view material in the context of a child).

To allow this fine control a flag would be needed that specifies how each context value should be merged with a matching one on a stored pattern's context. This provides a fine grained way for query behaviour to be specified.

4.4 Multiple Connections and First Class Bindings

The current implementation of Auld Linky permits an Association object to contain multiple members (represented by Binding objects), but in all other cases it only allows for single connections between objects.

returns, therefore FOHM would need to be extended to allow multiple connections at all points. Section 3.3 discussed a similar situation, where n ids could be attached at every point of the structure. Here we can only attach one id, or pattern, but it may resolve into many objects.

Multiple connections would have a negative impact on the pattern matching process as the server needs to attempt to match all possible permutations at any point where there is a collection of objects.

A further requirement is that structure servers should explicitly name all objects that undergo frequent revision. This would allow them to be versioned separately. FOHM currently defines Bindings as implicit objects wholly contained within Association objects. This presents some difficulties for FOHM versioning. Firstly, Bindings cannot be versioned directly. If clients want to version a specific Binding, then they have to version the whole Association object within which the Binding is contained. Secondly, the act of versioning an Association causes the creation of new revisions of all of the Bindings contained within it.

4.5 Maintaining the Version History

The problem with using pattern propagation to support versioning is that it does not record the version history of individual objects.

One solution is to store the version history implicitly within Context objects attached to each object. They will record the object's parent, children and its immediate siblings. This is a far from ideal solution. It is difficult to maintain and the role of Context objects is not to store meta-data about a structure, but to store meta-data that describes in which contexts that structure is visible.

A second option is to record the version history as an additional structure. This could even be stored in the same structure server as the evolving structure it is being used to record (this could take a similar form to the revision tree structure described in Sect. 3.2).

5 How Would Versioning Work in Practice?

In this section we describe how a contextual structure server implementing pattern propagation would be used to support the two core versioning operations of revision retrieval and revision creation.

5.1 Retrieving the Latest Version

One of the fundamental capabilities of a versioning system is to allow users to easily retrieve the latest version of a structure. It is also important to support naïve (non-versioning aware) clients by giving access to the latest version of a structure by default.

When using a contextual structure server to support versioning there is a problem in that naïve clients will query the server with a blank context object and thus match all versions (when what they really require is the latest version).

In Linky this is caused by the model of context matching. If a key is present in both contexts then it must have the same value to match but if it is only present in one context then it is assumed to match by default.

The way that Linky avoids this problem is to allows context values to be flagged as *required*, this reverses the normal matching behaviour and ensures that if only one of the context objects specifies a key then those context objects fail to match.

We could use this required flag within the linkbase on all object revisions save the latest one, this means that a user must specify a version key explicitly to retrieve anything other than the latest version.

It is also possible that with branching version structures there is more than one version available in a particular time slice and therefore more than one 'latest' version. It is up to the authors of such branches whether or not they use a similar mechanism to hide their versions from the view of naïve clients.

5.2 Retrieving an Earlier Version

Several approaches can be adopted for retrieving an earlier version of an object.

The first is to send a contextualised query that will match against the patterns stored in the Auld Linky structure server. This query will contain a context object that has a key that describes the versioning dimension that we are interested in, for example time. The Merge value will specify that we only want to match against stored patterns that are equal to the context value, such as FOHM objects that were created between 10:00am and 11:00am.

The second approach is to search for a specific object version by traversing the version history structure (such as the one shown in Fig. 2). This is only possible if the version histories of FOHM objects have been explicitly recorded as described in Sect. 4.5.

Both of these empower the *user* to select a particular revision. With pattern propagation it becomes possible for the *author* of the structure to choose a revision by connecting an object to their structure in the context of an earlier version.

5.3 Creating a New Version

To create a new revision a new object must be created in the structure server with the same id as the old revision. It is not necessary to change any existing connections. However, the structure server will have a way of denoting the latest revisions of an object and this must be updated to the new revision (for example, in Auld Linky the old revision must have its required flag set to true).

If a version history of the structure is being explicitly recorded it would also need to be updated.

6 Conclusion

In this paper our aim was to find out if a contextual model could be powerful enough to replace the selection engine of a versioning server. We approached this question by examining FOHM's contextual model as implemented within Auld Linky.

We found that FOHM does not have a powerful enough context model in its present form. This is because FOHM allows objects to be contextual but not the connections between them. By extending FOHM to include multiple

connections and context on those connections (via pattern propagation) FOHM could support versioning with context.

More generally this means that it is possible to reconcile context and versioning. General contextual models can be devised that are powerful enough to cope with the requirements of a versioning selection engine.

The extended contextual model is also more expressive when considering the non-versioning applications of context. It allows structures to be connected together in context. For example a link could be authored to a tour in the context of a child. When the link is retrieved it will include the children's version of that tour, even if the original query was made in the context of an adult.

By reconciling context and versioning we view a contextual structure server as a component that may slot into a versioning framework to take over the role of the selection engine. This results in an integrated approach to context and versioning where object revisions are manipulated in the same manner as other contextual views.

Other aspects of a versioning service (such as policy management and enforcement, object locking, run-time consistency etc.) can be provided by other components in the system framework, an architecture advocated by Component Based Open Hypermedia Systems (CB-OHSs) such as Construct [33,32]. If a version history is required then it could be explicitly maintained by a separate knowledgeable component (that could still use the structure server for storage).

In summary, we have shown that versioning is a specific kind of contextualisation and that it can be supported by a generic contextual model. We have presented an extension to FOHM that describes how this might be done in practice. It is our opinion that versioning is better seen as a specific use of context and that reconciling the two results in a more natural and consistent approach to both.

Acknowledgements

This research is funded in part by EPSRC IRC project "EQUATOR" GR/N15986/01 and by EPSRC MNA project "LinkMe" GR/M25919.

References

1. Anderson, K. M., Taylor, R. N. and Whitehead, E. J. 1994. Chimera: Hypertext for heterogeneous software environments. *Proceedings of the ACM European conference on Hypermedia technology (ECHT '94)* (Edinburgh, UK, Sep), 94-197.
2. Bailey, C., Hall, W., Millard, D. E., and Weal, M. J. 2002. A structural approach to adaptive hypermedia. *Second International Conference on Adaptive Hypermedia and Adaptive Web Based Systems* (Malaga, Spain).
3. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. 1994. The World Wide Web. *Communications of the ACM 37*(8), 76-82.
4. Davis, H., Reich, S., and Millard, D. 1997. A proposal for a common navigational hypertext protocol. Tech. report, Dept. of Electronics and Computer Science.

5. Fountain, A. M., Hall, W., Heath, I., and Davis, H. C. 1990. Microcosm: an open model for hypermedia with dynamic linking. *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)* (Rizk, A., Streitz, N., and Andr, J., Eds.), Cambridge University Press, 298-311.
6. Grønbæk, K. and Trigg, R. H. 1994. Design issues for a Dexter-based hypermedia system. *Communications of the ACM 37*(3) (Feb), 40-49.
7. Haake, A. and Hicks, D. 1996. Verse: Towards hypertext versioning styles. *Proceedings of the '96 ACM Conference on Hypertext* (Washingto, DC, Mar), 224-234.
8. Haake, A. 1994. Under CoVer: the implementation of a contextual version server for hypertext applications. *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology* (Edinburgh, UK, Sep), 81-93.
9. Haake, A. 1992. CoVer: a contextual version server for hypertext applications. *ECHT '92. Proceedings of the ACM conference on Hypertext* (Milan, Italy, Nov), 43-52.
10. Halasz, F. 1991. Seven issues: Revisited. The Hypertext '91 closing plenary. *Proceedings of the '91 ACM Conference on Hypertext* (San Antonio, TX, Dec).
11. Halasz, F. 1988. Reflections on NoteCards: seven issues for the next generation of hypermedia systems. *Communications of the ACM 31*(7), 836-852.
12. Hicks, D. L., Leggett, J. J., Nürnberg, P. J. and Schnase, J. L. 1998. A hypermedia version control framework. *ACM Transactions on Information Systems 16*(2), 127-160.
13. Marshall, C. C. and Shipman, F. M. 1997. Spatial hypertext and the practice of information triage. *Proceedings of the '97 ACM Conference on Hypertext* (Southampton, UK, Apr), 124-133.
14. Marshall, C. C. and Shipman, F. M. 1995. Spatial hypertext: Designing for change. *Communications of the ACM 38*, 88-97.
15. Melly, M. and Hall, W. 1995. Version control in Microcosm. *Proceedings of the Workshop on the Role of Version Control in CSCW* (Stockholm, Sweden, Sep).
16. Michaelides, D. T. and Millard, D. E. and Weal, M. J. and DeRoure, D. 2001. Auld Leaky: a contextual open hypermedia link server. *Hypermedia:Openness, Structural Awareness, and Adaptivity (Proceedings of OHS-7, SC-3 and AH-3*, Lecture Notes in Computer Science vol. 2266, Springer Verlag, Heidelberg, ISSN 0302-9743, 59-70.
17. Millard, D. and Davis, H. 2000. Navigating spaces: the semantics of cross domain interoperability. *Proceedings of OHS 6 and SC2*, Lecture Notes in Computer Science vol. 1903 (Reich, S., and Anderson, K. M., Eds.), Springer Verlag, Heidelberg, ISSN 0302-9743, 129-139.
18. Millard, D., Davis, H. and Moreau, L. 2000. Standardizing hypertext: Where next for OHP? *Proceedings of OHS 6 and SC2*, Lecture Notes in Computer Science vol. 1903 (Reich, S., and Anderson, K. M., Eds.), Springer Verlag, Heidelberg, ISSN 0302-9743, 3-12.
19. Millard, D. E., Moreau, L., Davis, H. C., and Reich, S. 2000. FOHM: A fundamental open hypertext model for investigating interoperability between hypertext domains. *Proceedings of the '00 ACM Conference on Hypertext* (San Antonio, TX, May), 93-102.
20. Nelson, T. H. 1987. Literary Machines, Mindful Press.
21. Nürnberg, P. J., Schneider, E. R. and Leggett, J. J. 1996. Designing digital libraries for the hyper-literate age. *Journal of Universal Computer Science 2*(9)
22. Reich, S., Wiil, U. K., Nürnberg, P. J., Davis, H. C., Grønbæk, K., Anderson, K. M., Millard, D. E., and Haake, J. M. 2000. Addressing interoperability in open hypermedia: the design of the Open Hypermedia Protocol. *New Review of Hypermedia and Multimedia*, 207-243.

23. Reinert, O., Bucka-Lassen, D., Pedersen, C. A. and Nürnberg, P. J. 1999. CAOS: a collaborative and open spatial structure service component with incremental spatial parsing. *Proceedings of the '99 ACM Conference on Hypertext* (Darmstadt, Germany, Feb), 49-50.
24. Schnase, J. L., Leggett, J. J., Hicks, D. L., Nürnberg, P. J., and Snchez, J. A. 1993. Design and implementation of the HB1 hyperbase management system. *Electronic Publishing—Origination Dissemination and Design 6*(1) (Jun), 35-63.
25. Slein, J., Vitali, F., Whitehead, Jr., E. J. and Durand, D. G. 1998. Requirements for distributed authoring and versioning on the World Wide Web. *ACM StandardView 5*(1), 17-24.
26. Soares, L. F. G., Filho, G. L. d. S., Rodrigues, R. F., and Muchaluat, D. 1999. Versioning support in the HyperProp system. *Multimedia Tools and Applications 8*(3), 325-339.
27. Van Dyke, P., H. 1993. Hypercubes grow on trees (and other observations from the land of hypersets). *Proceedings of the '93 ACM Conference on Hypertext* (Seattle, WA, Nov), 73-81.
28. Van Dyke, P., H. 1991. Don't link me in: Set-based hypermedia for taxonomic reasoning. *Proceedings of the '91 ACM Conference on Hypertext* (San Antonio, TX, Dec), 233-242.
29. Vitali, F. 1999. Versioning hypermedia. *ACM Computing Surveys 31*(4).
30. Whitehead, Jr., E. J. 2001. Design spaces for link and structure versioning. *Proceedings of the '01 ACM Conference on Hypertext* (Århus, Denmark, Aug), 195-204.
31. Whitehead, Jr., E. J. 2001. WebDAV and DeltaV: Collaborative authoring, versioning, and configuration management for the Web. *Proceedings of the '01 ACM Conference on Hypertext* (Århus, Denmark, Aug), 259-260.
32. Wiil, U. K., Hicks, D. L. and Nürnberg, P. J. 2001. Multiple open services: a new approach to service provision in open hypermedia systems. *Proceedings of the '01 ACM Conference on Hypertext* (Århus, Denmark, Aug), 83-92.
33. Wiil, U. K. and Nürnberg, P. J. 1999. Evolving hypermedia middleware services: Lessons and observations. *Proceedings of of the 1999 ACM Symposium on Applied Computing (SAC '99)* (San Antonio, TX, Feb), 427-436.
34. Wiil, U. K. and Leggett, J. J. 1996. The HyperDisco approach to open hypermedia systems. *Proceedings of the '96 ACM Conference on Hypertext* (Washington, DC, Mar), 140-148.
35. Østerbye, K. 1992. Structural and cognitive problems in providing version control for hypertext. *ECHT '92. Proceedings of the ACM conference on Hypertext* (Milan, Italy, Nov).

Metadata Usage in Multimedia Federations*

Mark Roantree and Damir Bećarević

Interoperable Systems Group
Dublin City University
Dublin 9, Ireland
{mark,damirb}@computing.dcu.ie

Abstract. There has been a large and varied amount of research into federated database systems, and more recently, this has advanced in the direction of federations of multimedia databases. However, in each project studied so far, none have placed special emphasis on the role of metadata in such systems. In this paper, a detailed specification of a multimedia metamodel suitable for a federation of databases, is provided. By illustrating the metamodel, it is clear to users, administrators and programmers exactly how data is structured, and the types of operations that are possible.

1 Introduction

Many applications require access to databases of unknown structure where the initial objective is to determine the database entities and their relationships prior to access or modification of data. In the area of federated databases, this requirement is stronger as a global administrator or engineer must take multiple heterogeneous software systems, determine similarities and differences across systems, and combine them to form one or more global schemas. A key support feature in this task is the presence of a schema repository where the database structure is described, and a metamodel which provides a set of legal relationships and actions for entities in the database.

In the EGTV project[16], the goal is to integrate a medium to large number of multimedia database systems which share a common metamodel interface. The contribution of this paper is in the specification of a metamodel for multimedia federations, of which an implementation has been mapped to both the object-oriented database standard (ODMG) and Object Relational (O-R) data models. Furthermore, there is an emphasis on the metadata aspects of integrating large multimedia systems. We argue that by specifying full details of the metamodel (or schema of the database repository), this enhances the integration process as it is clear how to interrogate the schema. The paper is structured as follows: in the remainder of this section we provide a brief overview of federated databases and the importance of metadata in the construction of federated schema; in Sect. 2 some related research is provided; in Sect. 3 a description of the system

* Supported by Enterprise Ireland Research Innovation Fund IF/2001/305

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 132–147, 2003.

architecture is provided; in Sect. 4 the multimedia metamodel is presented; in Sect. 5, an implementation is discussed; and finally in Sect. 6 we offer some conclusions.

1.1 Background and Motivation

The concept of a federation of databases [20] is one where heterogeneous databases (or information systems) can communicate with each other through an interface provided by a common data model. In our case, the common data model is the ODMG model [6], reengineered to remove redundancies, and extended to provide support for multimedia types.

The most common architecture for these systems is as follows: data resides in many (generally heterogeneous) information systems or databases; the schema of each Information System (IS) is translated to the format of the common model, and this new schema is called the component schema; view schemata are defined (subsets of the component schema) that are shared with other systems; these view schemata are exported to a global or federated server where they are integrated to form many global or federated schemata. Please refer to [15] for a more complete description of object-based federated database systems.

The key tasks which are simplified by the presence of a metamodel are in the manipulation of view schemas, the transportation of view schemas (between local and global servers), and the integration of local view schemas to form one or more federated (or global) schemas. In earlier work [18], we demonstrated how the metamodel could be used to describe and manipulate virtual entities which were defined in local ODMG databases. By using the metamodel to describe a subschema of virtual complex types, it was possible to transfer views (sub-schemas) between participating systems. Finally, the complexity of integrating object-oriented subschemas was shown to be reduced by having an ODMG metamodel to describe and alter virtual types.

2 Related Research

In this section we present an overview of research in the area of distributed multimedia systems. Issues addressed in these projects include multimedia data and metadata representation, query languages for multimedia and federated multimedia architectures.

2.1 SQL/MM Still Image Standard

The SQL/MM Still Image Standard [21] provides SQL multimedia extensions for manipulation of still images stored in the database. The standard defines a `SI_StillImage` data type for representation of still images. `SI_StillImage` stores image data as BLOBs (Binary Large Object), while image parameters

(height, width, etc.) are stored as text values. Each SI_StillImage object type defines constructors and methods for basic image manipulation. Additional methods for context querying based on average colour, colour histogram, positional colour, and texture are also defined in standard.

The main advantage of SQL/MM as a query language for multimedia is that it provides multimedia extensions for the existing database query language SQL, and as such, users need not learn a new language for multimedia transactions. The disadvantage of this approach is that SQL/MM is limited only to relational and object-relational databases that support SQL. Furthermore, the SQL/MM standard does not support any other multimedia type except still images. Data distribution features and a multimedia repository schema are not included in the standard.

2.2 The Garlic Project

Garlic [5] is a distributed database system for storage and manipulation of heterogeneous multimedia data developed at the IBM Almaden Research Centre. It provides a global schema for multimedia data originating from different data stores, where a data store can be any database or specialized multimedia storage system. A global schema is a union of local schemas transformed to the Garlic data model. Data stores are connected to the Garlic system by wrappers [7] that transform the proprietary data model to the data model of the Garlic global schema. The Garlic data model is based on the ODMG model but is extended for multimedia support. The main extension to the ODMG model is the concept of views. The views are defined on top of multimedia classes in the Garlic global schema. Views can extend, simplify or reshape class properties and methods. The limitation of the view system is that each view is formed from a single base class. Objects originated from different data stores can be combined using "complex objects". Complex objects are stored in the special Complex Object Repository and they model relationships that exist between multiple multimedia objects. The query language for Garlic is based on SQL and extended with object-oriented features like references, collections and operations. These extensions include predicates and operations for context querying of multimedia objects. A global query is decomposed to set of smaller queries that are executed on the wrapper databases. No description of metamodel is described in the published work on Garlic.

2.3 Multimedia News-on-Demand System

The news-on-demand system [14] is as a multimedia project developed at the University of Alberta in Canada. Each news document consists of textual and multimedia elements with spatial and temporal relationships. The spatial relationships between multimedia elements are represented in SGML, while the Hy/Time standard was used for temporal relationships. From the database aspect, the system provides an object-oriented representation for multimedia elements. Non-continuous media (text and still images) are stored in the object

database, while continuous media (audio and video) are stored in a special media file server. The architecture does not provide support for data distribution, since all objects must be stored in these two data stores.

Data types for multimedia are defined as a class hierarchy. The base multimedia type is abstract class `Atomic` and all other types in the class hierarchy are descendants of `Atomic`. Classes that represents single multimedia (like `jpg`, `avi`, `mpeg`) derive from leaf classes in the class hierarchy. Complex multimedia objects consist of interrelated single multimedia objects. Complex (data) objects are represented as SGML and HyTime documents, where SGML describes spatial relationships between component single multimedia objects, and HyTime models temporal relationships. For each SGML DTD definition, a single class is created in the object database. A query language for multimedia was also developed as a part of the project. The language is based on ODMG OQL, and extended with multimedia features (MOQL). These feature include functions and predicates for spatial and temporal querying of multimedia data. However, the language does not have transaction or multimedia manipulation capabilities. A detailed specification of the MOQL language can be found in [12].

2.4 Multimedia Federations

The Federated Multimedia Database System [3] comprises textual and multimedia data into a federation which maintains a single global schema. Local data stores connect to the system through wrappers that provide standardized interfaces to local data. A global federated schema is constructed in two steps. First, two intermediary schemas are constructed containing *media* and *structured* (non multimedia) data schema. A structured data schema is a union of all schemas exported from the databases that store regular non multimedia data, while the media schema represents a union of the all schemas exported from the local media stores. The media schema does not physically store multimedia objects, but contains proxies that map to real multimedia data. Classes defined in each local media schema are organized in the multimedia class hierarchy. At the top of the class hierarchy are `TObject` and `TSingleMedia` classes. `TObject` is base class for all classes in the schema while `TSingleMedia` is the base class for all proxy media objects. Audio, video and image classes must derive from this class hierarchy. In the second step of global schema construction, multimedia and structured data schemas are integrated in the global schema and stored in a special repository called *internal database*. The integration is accomplished by establishing relationships between multimedia and structured classes. New complex multimedia classes can be constructed in the global schema. These classes integrate one or more singe multimedia classes and include spatial and temporal relationships between them. All complex classes that require spatial and temporal features must derive from the `TVisualElement` base class.

2.5 The Fischlár System

Fischlár [13] is a multimedia system for digital video indexing and browsing. The system enables clients to record TV programs or watch previously recorded videos through the web interface. Selected TV programs are recorded in an MPEG-1 encoding format and stored in a regular file system. Recorded videos are then processed by video indexing software that performs shot and scene boundary detection. This information is used for the generation of web pages with thumbnail images representing the beginning of each scene within the video. Clients can select any of these images to begin video playback starting from that particular scene. Real-time video streams are then broadcasted over the TCP/IP network from the video-streaming server to clients. A streaming server (Oracle Video Server) stores MPEG-1 video files in a specialized file system optimized for fast retrieval, while scene and shot indexing information are fully contained in the web interface.

2.6 Summary

While this section demonstrates the wide range of research projects in both multimedia, and in some cases, federated multimedia systems, there is little or no emphasis on the subject of metadata. By specifying (and publishing) a detailed metamodel, it provides clarity as to the structural and possibly behavioural capabilities of the system. When integrating heterogeneous systems, this extra power is crucial.

3 Operational Architecture

In this section, a brief overview of the operational architecture illustrated in Fig. 1 is provided. There are three broad levels in the EGTV architecture, although the structural architecture has five levels (similar to [20]), but this paper does not address issues of schema export and integration. Instead, this section will focus on the role of the new metamodel interface for database creation and manipulation.

In the federation, all data is stored in either ODMG compatible or object-relational databases at the *Database Layer*. The Database Layer is used for physical storage of data and multimedia objects. However, the creation of database schemas, and the manipulation of data is not performed at this local level, but at the *Metamodel Layer*. This removes problems of heterogeneity when dealing with different database types.

A common interface to each storage device is supplied by the EGTV metamodel as the *Metamodel Layer* in the architecture. This provides the illusion that all local storage devices are identical, and this is later exploited when constructing global schemas. The metamodel is extended to include multimedia types

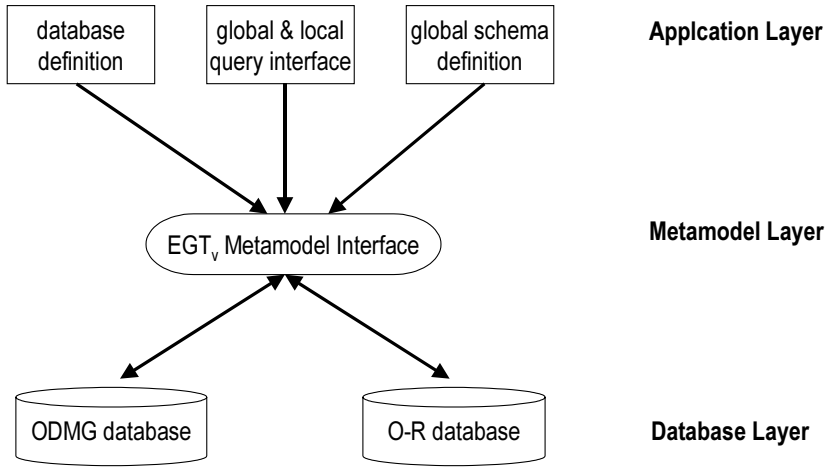


Fig. 1. Operational Architecture

and is mapped to both ODMG and Object-Relational[1] models. All applications interact with data through this layer.

At the *Application Layer*, there are three processors which provide functionality for building and manipulating both single database server and federated database applications. Databases can be defined using either an extended OQL or an XML interface; queries can also use either ODL or XML and be local or distributed; and a final processor facilitates the construction of federated or global schemas. For a more complete description of these processors, please refer to [4].

4 Metamodel Specification

This section describes the object-oriented metamodel designed for multimedia databases, and with a special emphasis on the integration process where multiple heterogeneous multimedia databases are combined. In this context, the metamodel defines an object-oriented meta-schema for representing textual and multimedia metadata for databases at both the component and federated layers. For space reasons, we limit the discussion of each subset of the metamodel to a brief overview: please refer to [1] for a more detailed description, including a breakdown of the ODMG metaclasses which have been eliminated or amended, and those which are new.

Databases at the component layer of the architecture use either an object-oriented or object-relational data model. The metamodel is based on the generic metadata definition for ODMG compliant databases as described [6] and the

[1] We treat the Oracle 9i model as a standard as it has the most advanced object-relational model.

C++ implementation defined in [10]. Our metamodel eliminates ambiguities and redundancies present in both specifications by flattening the metamodel structure, and significantly reducing the overall complexity. Some modifications required for the representation of multimedia data types and behaviour were also introduced. Our metamodel does not incorporate metadata access interface (as the ODMG metamodel does) because this limits generic query capabilities [9]. The structure of our metamodel is described in the following subsections, and a detailed description is provided for all metaclasses that differ from the ODMG metamodel specification. Each metaclass is classified to one of three groups: new metaclasses, modified metaclasses and reused metaclasses. Modified metaclasses exist in the ODMG metamodel, but their structure is redefined in our metamodel. All metaclasses in our metamodel are prefixed with the `sys` prefix.

4.1 Defining Name Scopes

Each database entity has a name that must be unique within the scope to which it belongs. For example, attribute names are uniquely defined within the containing class, while class names are unique in the database schema. Name scopes and containment relationships in the metamodel are closely related because each metaclass provides a scope for all its contained elements. The ODMG metamodel separates name scopes and containment definitions into two different relationships, introducing redundancy to the metamodel. For example, the ODMG `d_Class` metaclass has two separately defined relationships to the `d_Attribute` metaclass: scope relationship and containment relationship. Our metamodel eliminates that design ambiguity by defining a single containment/scope relationship between metaclasses as illustrated in Fig. 2.

The `sys_MetaObject` is an abstraction for all elements in the metamodel and defines common attributes such as `name`, metaclass type (`metaType`) and `comment`. These three attributes provide name, type and user defined comment properties for all elements stored in the metamodel. Metaclasses that are not capable of containing other metamodel elements, like `sys_Property`, `sys_Parameter` and, `sys_Inheritance` derive directly from the `sys_MetaObject`, while all container metaclasses derive from `sys_ScopedObject`. The `sys_ScopedObject` defines the containment relationship (`contains`) to `sys_MetaObject`, so that each instance of `sys_ScopedObject` can contain and define name scope for many `sys_MetaObject` instances. For example, a class which is an instance of the metaclass `sys_ScopedObject`, can contain attributes, relationships and operations which are uniquely identified within the scope of that class.

Since the `sys_ScopedObject` metaclass also derives from the `sys_MetaObject` metaclass, each container metaclass can recursively contain another container class. Metaclasses `sys_Class`, `sys_Schema` and `sys_Operation` derive from the `sys_ScopedObject`, as they provide naming scope for elements contained within them. The top level scope is the database schema itself (`sys_Schema`), and it contains the classes (`sys_Class`) defined by users.

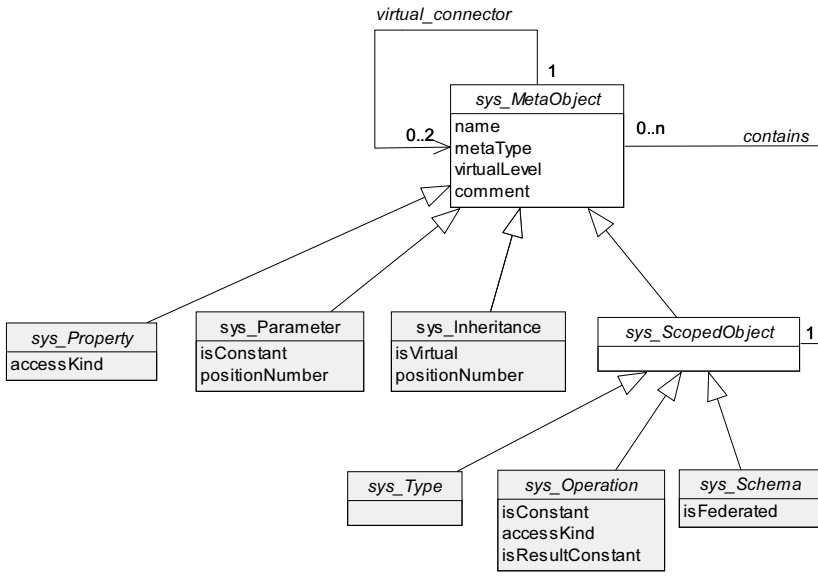


Fig. 2. Metadata Definition of Name Scopes

4.2 Defining Types

Type metaclasses provide a description for types defined in the database. All data types are represented in a metadata class hierarchy as illustrated in Fig. 3. The `sys_Type` metaclass is an abstraction for all types in the database, and the more specific metaclasses which derive from it. Our metamodel is enhanced by permitting operations for the `sys_Type` metaclass, whereas the ODMG metamodel permits only user defined classes to have operations. Moving operation support to the level of `the sys_Type` base class enables the definition of operations not only for classes, but also for other data types (e.g. multimedia and collection types). The importance of this feature arises from the fact that the internal structure of complex data types is fully encapsulated and the only interface is provided through operations. For example, a `jpegImage` multimedia data type can have operations for query by pattern, resizing and rotating of an image it contains. These operations are registered in the metamodel and publicly available, while the implementation and storage details of the `jpegImage` data type are hidden from the user.

The system and user defined types are represented as a specialisation of the `sys_Type` metaclass. System types are used as attributes of classes, or parameters of operations, and they cannot be instantiated to persistent self contained database objects. System types can achieve persistence only as attributes of user defined classes. All system types can be classified as primitive and collection types.

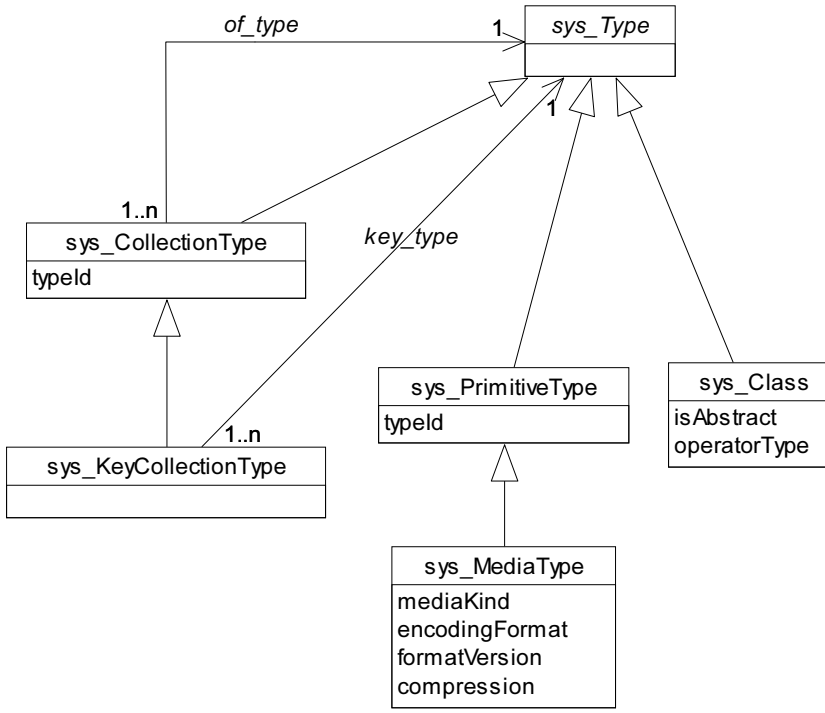


Fig. 3. Metadata Definition of Data Types

- **Primitive types:** Represented in the `sys_PrimitiveType` metaclass. The full set of allowable types are `Integer`, `Float`, `String`, `Date`, `Boolean` and `Octet`. Special data types for multimedia storage are incorporated into this metamodel and they are represented as instances of the `sys_MediaType` metaclass. The metamodel defines the `sys_MediaType` metaclass as a specialisation of the `sys_PrimitiveType`. The `mediaKind` property identifies multimedia type (`audio`, `video`, `text` or `image`), while `encodingFormat`, `formatVersion` and `compression` provide information on media encoding characteristics.
- **Collection types:** Collections store multiple objects of one system type and are represented by the `sys_CollectionType` metaclass. Supported collection types include `Set`, `List` and `Array` as specified in the ODMG standard.

4.3 Defining Properties

The `sys_Property` metaclass is an abstraction for all property types that can be specialised as attributes or relationships. Members of a class are specified in the `sys_Attribute` metaclass, where each attribute can be optionally defined

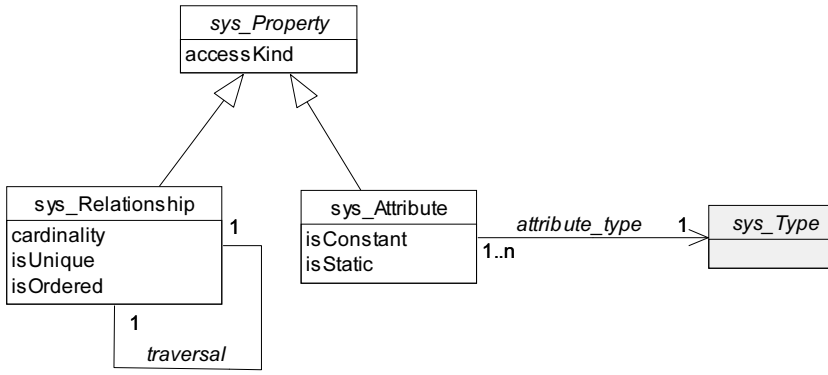


Fig. 4. Metadata Definition of Properties

as static or constant using `isStatic` and `isConstant` properties. Attributes can be of a system or a class type, where each attribute has only one type, while one type can be used by many attributes. This is represented with the `attribute_type` relationship between `sys_Attribute` and `sys_Type` metaclasses where the `sys_Type` is a superclass for all types in the metamodel.

The `sys_Relationship` metaclass defines bidirectional relationships between two classes where a cardinality of `one` or `many` is specified for each side of the relationship. The traversal property of the `sys_Relationship` returns the other side of the bidirectional relationship. Each relationship with cardinality greater than one can have ordered values (`isOrdered` property) or it can be defined as unique (`isUnique` property).

4.4 Defining Inheritance

Inheritance relationships are defined for classes only. Inheritance in the metamodel is represented by the `sys_Inheritance` metaclass. The instance of the `sys_Inheritance` metaclass has an `inherits_to` and a `derives_from` relation-

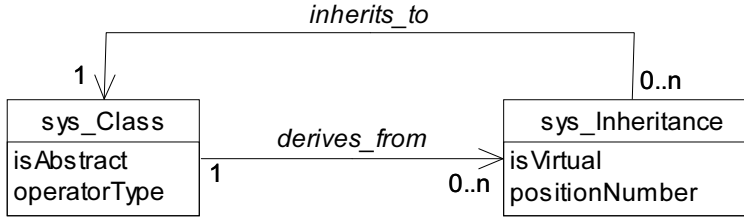


Fig. 5. Metadata Definition of Inheritance

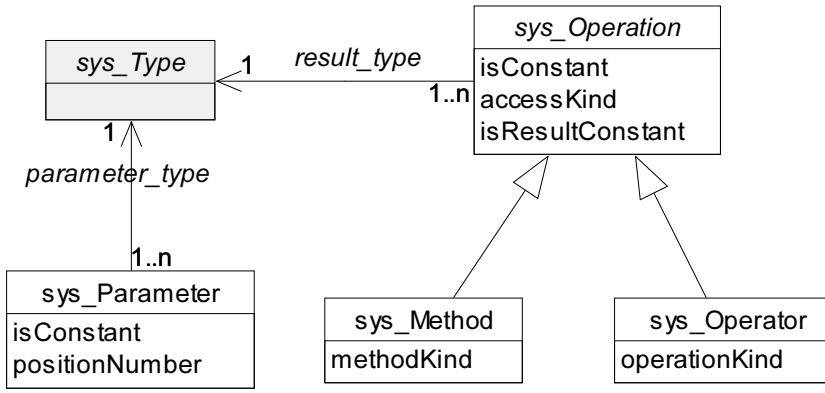


Fig. 6. Metadata Definition of Operations

ship with two instances of `sys_Class`. Each class has a list of inherited classes and a list of derived classes. The `positionNumber` parameter indicates the order of base classes in multiple inheritance definitions.

4.5 Defining Operations

Operations can be defined for both system types and classes. Operations specified for system types are part of the type definition and cannot be modified by the user, while operations on classes are user defined. The `sys_Operation` metaclass is an abstraction for all operations defined in the database. There are many issues concerning representation and invocation of database behaviour, but they are not be addressed here as they form part of a separate ongoing project [11].

4.6 Defining Schemas

A schema represents the top level container for classes and object views. The ODMG implements a schema as an instance of `the d_Module` metaclass, while in our metamodel `sys_Schema` is an abstraction for database schema and view subschema metaclasses. An instance of `sys_DatabaseSchema` represents one database schema, defining a global scope for the database objects it contains. Objects that can be registered within a database schema are base classes and views.

4.7 Defining Views

The `sys_SubSchema` is a special "schema" for representing object views. It contains base and virtual classes and each subschema belongs to one database

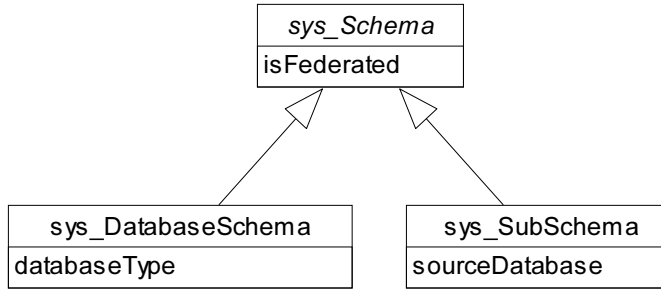


Fig. 7. Metadata Definition of Schema

schema. This representation is necessary as a view is always based on a subschema, and not on a single virtual class. Object views provide schema restructuring functionality for object-oriented database models. This feature is commonly used in federated database systems for the construction of different component and federated schemas. Object view support is not provided in the ODMG standard, but the EGTV metamodel defines extensions for representing view metadata. View support is designed to represent the view mechanism specified in [18]. Each object view, represented as an instance of the `sys_SubSchema` metaclass, contains one or more virtual classes. A virtual class is recursively constructed from base classes or other virtual classes using restructuring operators specified in [18].

4.8 A Meta-metadata Level

Each metadata model describes the structure of the database schema at some level of abstraction. Our metadata model is specifically constructed to support multimedia metadata by recognising multimedia types as a special form of data

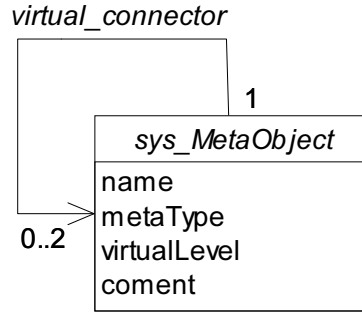


Fig. 8. Metadata Definition of Views

type. The model in which the metamodel is specified and constructed is called the meta-metadata model. Metadata models for representation of specific database models (e.g. multimedia) can be easily defined in the meta-metadata model. Migration from the one metamodel structure to the another is accomplished by changing metamodel representation in the meta-metadata model. Our specification of meta-metadata model is illustrated in Fig. 9. We recognise the meta-metadata model as beneficial to our system, because it will allow us to specify new metamodels and to add new metaclasses to the existing ones.

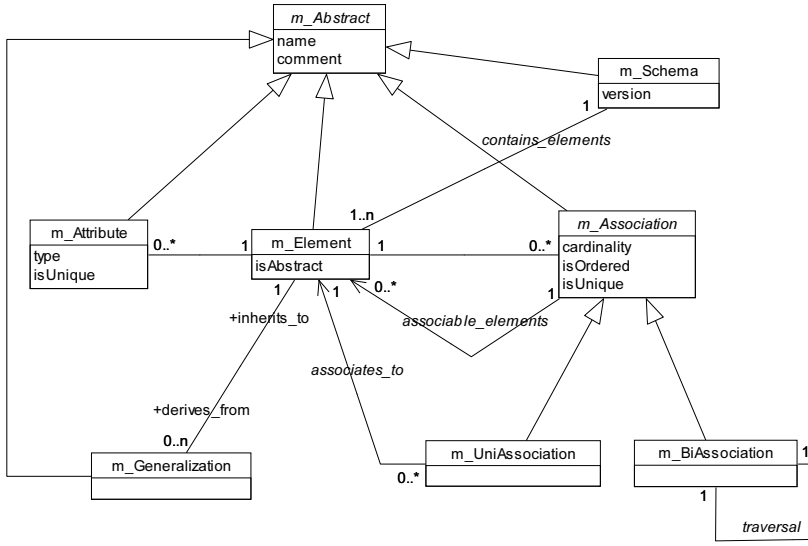


Fig. 9. The Meta-Metamodel

The `m_Abstract` metaclass is an abstraction for any type of metamodel element. It defines `name` and `comment` attributes common for all entities in the meta-metadata model. The `m_Abstract` can be realised as element, attribute, association, generalisation or schema. The `m_Element` metaclass represents a general container element, and instances of the `m_Element` correspond to `sys_Class` metaclasses in the metamodel definition. Each `m_Element` can contain attributes and associations represented by the `m_Attribute` and `m_Association`. The `type` defines attribute type, while `isUnique` property specifies if attribute instances must be unique. Associations in the meta-metadata model can be unidirectional (`m_UniAssociation`) or bidirectional (`m_BiAssociation`). The `cardinality`, `isUnique` and `isOrdered` properties are defined for both association types, but only bidirectional associations have a `traversal` link to the inverse association element. Inheritance relationships between metadata elements are repre-

sented by the `m_Generalisation` metaclass, while the `m_Schema` is the root container for all elements in the meta-metadata model. Each `m_Schema` instance corresponds to one metamodel schema defined in the meta-metadata model. The `associable_elements` relationship between `m_Association` and `m_Element` metaclasses defines association rules for metamodel instances of the meta-metadata model. The relationship specifies all subclasses to which an association defined between their superclasses, can be propagated. This feature enables definition of strict association rules that specify which subclasses are allowed to create an association link defined for their superclasses.

5 Implementation

A working prototype has been implemented for Versant 6.0 (ODMG) and Oracle 9i (object-relational) databases. It was developed and tested on Windows platforms (Windows 2000 and Windows XP), but a Linux version is due to complete shortly. The metamodel implementation for Versant is specified using *C++* header files, built and stored (as metaclasses) in the database using the Versant schema import utility. This process creates an extended schema repository which is an implementation of the multimedia metamodel. The Oracle implementation of the metamodel is developed as a SQL-99 script file containing SQL DDL statements for creating object types and object tables. This SQL script is executed to create a schema repository, implemented as a set of object-relational tables constructed from object type definitions.

A user-defined multimedia schema can be specified using a new object definition language, ODL_{X}[2], which is an XML-based schema definition language. This was developed for representing object-oriented multimedia database schemas in a database and platform independent format. The XML Schema is used as a specification language in which the ODL_{X} is defined. The specification language was created and validated for correctness using XML Spy 4.3, a development tool for XML. The Schema Generator program, developed as a part of the project, parses ODL_{X} file, and creates metadata and schema definitions in the target database. The program is written in C++ using Microsoft Visual Studio 7.0 and Apache XERCES libraries for XML. The XERCES DOM parser validates the ODL_{X} file against the XML Schema specification and constructs object representation of the ODL_{X} file. The Schema Generator uses Versant C++ libraries as a query and update interface for Versant database, while Oracle OCCI interface is used for Oracle database access.

By implementing a prototype, we are now in a position to test both the power of the metamodel (in terms of building multimedia databases and applications), and the possibility of extending personal metamodels to introduce new types and operations (using the meta-metamodel layer). Presently, users can define and query multimedia databases, while current research is focused on the development of federated multimedia views.

6 Conclusions

In this paper we illustrated our approach to designing and implementing a metamodel for multimedia databases. The goal of this research is to provide a standard metamodel interface to object-based multimedia systems, while the contribution is in the fact that no federated database research project (covered by this group) has published their metamodel. By publishing the metamodel details, it is possible to create an architecture which is both open and extensible. We feel that this will prove to be beneficial when creating federations of multimedia databases, which forms part of our current research focus.

References

1. Bećarević, D. 2002. A metadata model for a multimedia database federation. *Technical Report No. ISG-02-01*, Dublin City University.
2. Bećarević, D. 2002. An XML object definition language. *Technical Report No. ISG-02-04*, Dublin City University.
3. Berthold, H. and Meyer-Wegener, K. 2001. Schema design and query processing in a federated multimedia database system. *Proceedings of Cooperative Information Systems, 9th International Conference*, Springer.
4. Bećarević, D. and Roantree M. 2002. A Metadata Approach to Multimedia Database Federations. Submitted for publication to the IST Journal.
5. Carey, M., Haas, P., Schwarz, P., Arya, M., Cody, W., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., Thomas, J., Williams, E. and Wimmers, L. 1995. Towards heterogeneous multimedia information systems: the Garlic approach. *Proceedings of the RIDE-DOM '95,* IEEE-CS, 124-131.
6. Cattel, R., and Barry, D. (Eds.). 1999. *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.
7. Hass, L., Miller, R., Niswonger, B., Tork, R. M., Schwarz, P., Wimmers E. 1999. Transforming heterogeneous data with database middleware: Beyond integration. *IEEE Data Engineering Bulletin 22*, 31-36.
8. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW). 2001. *Proceedings of the 9. GI-Fachtagung*, Springer.
9. Habela, P., Roantree, M., and Subieta, K 2002. Flattening the metamodel for object databases. *6th East European Conference on Advances in Databases and Information Systems* (ADBIS), Springer (to appear).
10. Jordan, D., 1998. *C++ Object Databases: Programming with the ODMG Standard*, Addison Wesley.
11. Kambur, D. and Roantree, M. 2001. Using stored behaviour in object-oriented databases. *Proceedings of 4th International Workshop Engineering on Federated Information Systems* (EFIS 2001), Infix-Verlag.
12. Li, J, Ozsu, T., Szafron, D, and Oria, V. 1997. Moql: a multimedia object query language. *Technical Report TR-97-01*, Department of Computing Science, University of Alberta.
13. Lee, H., Smeaton, A., O'Toole, C., Murphy, N., Marlow, S., and O'Connor, N. 2000. The Físchlár digital video recording, analysis, and browsing system. *Proceedings of the RIAO 2000 - Content-based Multimedia Information Access.*

14. Ozsu, T., Szafron, D., El-Medani, G., El-Medani, S., Iglinski, P., Schoene, M., and Vittal, C. 1996. Database management support for a news-on-demand application. *Proceedings of The International Society for Optical Engineering 2617*, 24-3.
15. Pitoura, E., Bukhres, O., and Elmagarmid, A. 1995. Object orientation in multidatabase systems. *ACM Computing Surveys 27*(2), 141-195.
16. Roantree, M. 2002. Efficient global transactions for video media. *Technical Report No. ISG-02-03*, Dublin City University.
17. Roantree, M. 2002. Metadata management in federated multimedia systems. *Australian Computer Science Communications 24*(2), 147-155.
18. Roantree, M., Kennedy, J., and Barclay, P. 2001. Constructing view schemata using an extended ODL. *9th International IFCIS Conference on Cooperative Information Systems (CoopIS)*, LNCS 2172, Springer, 150-162.
19. Roantree, M., Kennedy, J., and Barclay, P. 2001. Using a metadata software layer in information systems integration. *3rd Conference on Advanced Information Systems Engineering (CaiSE)*, LNCS 2068, Springer, 299-314.
20. Sheth, A. and Larson, J. 1990. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Computing Surveys 22*(3), ACM Press.
21. Stolze, K. 2001. SQL/MM Part 5: Still Image *The Standard and Implementation Aspects*. In [8], 345-363.

Data Mining Using Links in Open Hypermedia

Dragos Arotaritei and Peter J. Nürnberg

Department of Computer Science
Aalborg University Esbjerg
Niels Bohrs Vej 8
6700 Esbjerg, Denmark
{dragos, pnuern}@cs.aue.auc.dk

Abstract. We present a survey of data mining using web links. The XLink standard provides new possibilities to mine the web but also poses complex new problems. In this paper, we analyze the new challenges posed by the Xlink standard and propose a model to mine XLink information on the web. Our model combines local and global information in a distributed web environment along with a dynamic approach for XLink paths in separated documents.

1 Introduction

The World Wide Web has become a huge source of data and information, and continues to grow very quickly. Information retrieval and new knowledge discovery under these conditions becomes increasingly difficult.

Many applications deal with various aspects of data mining problems [12] on the Web. A common taxonomy of data mining (DM) tasks describes three basic types of mining: web content mining; web usage mining; and, web structure mining. Some of mining applications share common aspects that belong to several of these categories.

Among the useful sources of valuable information used to discover knowledge from the Web, link analysis especially has received an increasing amount of interest in the last several years. Links in or to a document represent a valuable source of pre-processed knowledge or association that are created by the author of the Web document containing these links.

The rest of the paper is organized as follows. In the Sect. 2, we present a survey of the work in the area of link analysis and DM using links. In Sect. 3, we analyze and identify the problems in the new context of the XLink standard. A model to deal with DM tasks is proposed in Sect. 4. The conclusions and direction of future research are present in the Sect. 5, including some preliminary work involving Bayesian Networks.

2 Survey of Related Work

Many papers represent the Web as a graph [14]. Different aspects can be mined if we refer to HTML and XML pages. Note that all the papers mentioned in our

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 148–154, 2003.

survey refer to embedded links. (See below for a complete definition of embedded versusexternal links.)

The structure of HTML is mined using templates in [13]. The DTD schema is discovered from a collection of XML documents using a systematic approach improved by six heuristic rules described in[16]. A graph description language XGMML and a web-log report description language LOGML (both XML applications) are used in[21] and [22] for a mining algorithm on the Web, using frequent pattern mining frequent sets, frequent sequences,and frequent trees.

Web surfed paths are subject of user mining taxonomy. Nodes model the visited pages and the clickstreams of web surfers are modeled by a mixture of hidden Markov models in [26]. The model is a mixture of hidden Markov models and additional data trained using EM algorithm. Static user data and prior information (prior knowledge of the dynamics) are incorporated in algorithm [26]. The probabilistic link prediction along with path analysis is presented in [23]. Information extracted from log files from a web server; adaptive navigation and the sequence of states (URLs) followed by a user in the browsing process (a "tour"); and, personalization of hub/authority are described in [23].

Extensive research about the capabilities of categorization of hypertexts using different classifiers (Naive Bayes, Nearest Neighbor, and Foil) is made in the paper [9]. Some regularity properties generated from metadata and co-referencing are investigated [9].

Both the problems of data mining and information retrieval by using a multiagent system are subject of [15]. The users send queries in Prolog-style and the evaluation of the queries is made in an intelligent co-operative mode in a distributed environment [15].

A different approach using the queries is proposed in [8]. A query is composed of a URL of a page – a search engine produces as output a set of related web pages. Authors define the related page as a page that has the same topic [8]. Two algorithms are described in [8]: the Companion algorithms derived from HITS algorithm proposed by Kleinberg, and the Cocitation algorithm (frequently pages co–cited with the input URL are identified).

The classification problem for indexing purposes for unstructured hypertext databases is discussed in [5]. A robust statistical model that works with the metadata extracted from the neighborhood of the link information from documents is proposed. Experiments with text in different combinations among local neighbors and tags are described along with an iterative relaxation label technique using conditional probabilities [5].

The graph of linkage between sites is studied in [2]. A weighted graph is used to model the hyperlinks among pages of the hosts and the linkage between hosts. Two techniques to mine the related web hosts are used: link frequency and co–citation analysis [2]. The related hosts are discovered using a proposed score formula that has been applied for a experimental number of 100 random hosts[2].

A distributed approach from web surfing logs collected from a distributed heterogeneous host is shown in [7]. The authors used two types of Bayesian

Networks (BN) – local and central – in order to combine them and to form a collective BN modeling all the data. The authors used three datasets to test the collective Bayesian learning: ASIA model; real web log data; and, simulated web data [7].

The techniques based by the HITS algorithm (Hyperlink-Induced Topic Research) are used in [10] to analyze the web communities using a notion of hyperlinked commu-nities on the web. Communities are seen as a container of "authoritative pages" meanwhile "hub pages" link the pages. Briefly, the HITS algorithm is based on iterative process that assigns two weights for a page (authoritative and hub) according to numbers of hyperlinks from the current page to a another page.

Different problems of DM using basically the technologies based on HITS are depicted in [6]. Communities are discovered using a subgraph-enumeration technique, named *trawling*. The link information can be combined by content using heuristic rules, and the taxonomies can be constructed in a semi-automatic manner. The classification of the web pages are tested using Markov Random fields in am embedded system called HyperClas [6].

The performance of the HITS and Google PageRank algorithm under small perturbations is analyzed in [17]. As result of stability analysis, two new algorithms were proposed in [17]. Randomized HITS use a random-walk based algorithm in the basic idea of HITS and the stationary distribution on some steps as authoritative and hub weights. The subspace HITS stabilize the subspace spanned by eigenvector instead of individual eigenvector of the matrix that can unstable [17].

A probabilistic relational model is extended to be used in a unified probabilistic system that comprises both textual content and links structures from a collection of documents is proposed in [24]. The Probabilistic relational model is extended from relational schema and a probabilistic model for attributes to the Web in order to be used for prediction by belief propagation method [24].

3 Analysis of DM Using XLinks

Hypermedia (HM) concerns the representation of manipulation of explicit structure [18]. HM is often used in applications in which users want to mark relationships among arbitrary data as semantically meaningful. Many HM systems present users with associative structures, such as navigational links (e.g.,[25], but the general principles underlying HM structure management are quite general, having been used to support applications as diverse as software engineering [1], botanical taxonomic classification [19], and workflow management [11].

Recently, there has been a great deal of interest in DM applications on the World Wide Web (WWW). The WWW is the most widely known example of an HM system. Many approaches focus on augmenting traditional DM algorithms with the extra semantic information represented in the HM links found in or to many WWW documents. However, to date, these augmentations have focused on only one type of WWW link – namely, the embedded link.

Embedded links, as the name implies, are a physical part of the document from which the link originates. (One particularly common example of an embedded link on the WWW is the <A HREF="some-url">text</A> tag inserted into an HTML document.) The current DM focus on embedded links is understandable, as they constitute the vast majority of links in use today.

However, there has been increasing interest in supporting external links on the WWW. The notion of external links has been an important research issue in the open HM community since the late 1980's (see, e.g., [20]). Currently, the WWW Consortium (or W3C, the main standards body for the WWW) has endorsed a standard representation for external links on the WWW, called XLink [27]. Already, Netscape 6, a popular WWW browser, supports the XLink standard, as do several other systems.

DM in open HM (i.e., in the presence of external links) presents a variety of complications to the traditional WWW/DM approaches.

The links from documents can be stored in a separate document. Even this seems at first sight only a slight complication. Practically, this means that that local documents can have corresponding sets of links in one or more remote locations, requiring DM algorithms to be applied in an distributed environment.

An associative "link" can in fact be a sequence of links, one pointing to another to another, etc., forming a possibly very long chain. We may be able to confer upon these link chains some dynamic properties (like mining sequential paths).

Metadata can be associated and embedded with links, so we can exploit also this level of information or order to mine the links.

4 A Mixed Model for DM in Open Hypermedia

The links that are included in a document can be located in a separate document. Practically, a central server can collect all the documents containing links and analyze, rearrange and mine useful information.

However, from the links used in documents and citation that surround the close neighborhood of a document, it is often difficult to determine the importance of individual links in a path structure. One reasonable assumption is that the links are evaluated with the same importance link, somewhat similar to the attributes in a basket market used by DM algorithms for the association rule. So, this part is susceptible to be modeled by Markov models if we consider the graphical modes as basis for our representation.

However, the entire set of the links can be considered as a set of multi-valued attributes in a directed graph, suitable to be modeled by a Bayesian Network. On the other hand, because of the inherent distribution present in this problem, it is easier to use local models of Bayesian Networks for the links above in order to combine and to obtain a global model.

An important point is the message passing between local networks and central authority that models the global probabilistic network. Because of our implemen-

tation is primary focused on Java, we use RMI in order to obtain probabilistic information from each node to calculate the dependent probability on each node.

Some additional problems must be taken into account. The basic problems that must be able to solved by our proposed architecture are:

- incomplete information (e.g., server down, temporarily unavailable, etc.);
- incorrect information (bad or obsolete links); and,
- progressively outdated info (the info referred to by links are updated during the mining process.)

Associated possible solutions are:

- simply ignore the link or predict by combinatory expansion;
- ignore or use the cache; and,
- incrementally update (i.e., use "snapshots").

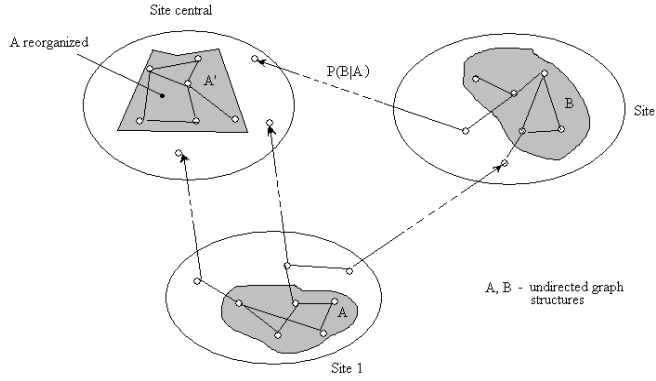


Fig. 1. The mixed model by directed and undirected graph

In the figure above we represented the main components of our proposed architecture. The conditional probability calculation for directed and undirected probabilistic graph models is based on those described in [4].

The service for client is implemented as a service that may be added to the Netscape browser.

Two categories of problems are taken into account at this stage of development: mining the community of interest (that is communities of related topics); and, scholarly recommendation system for scientific literature.

We remark that in the last case, some topics are not included in our approach because of major difficulties in preprocessing stage (consuming time). A relevant topic in this category is mathematical papers that present difficulties to extract information without many heuristic rules.

5 Conclusions

We presented a survey of the research in the area of data mining using links. The main techniques and associations with other technologies in order to improve link mining are analyzed.

The problematic of the new standard Xlink is analyzed in the data mining framework. A hybrid complex model based on distributed belief networks [3,4] and dynamic analysis of the external links is proposed.

Because of potential complexity of the external links in open hypermedia, that is a composed link can have many individual links, an approximate solution of solving the local inference to integrate in global inference is taken into account.

Metadata associated to links, can play a role in estimation of *a priori* probabilities using classic *a priori* data mining algorithms similar to the basket market model [12].

Because external links may be stored in a remote document of links, the co-citation analysis is more difficult in the XLink and OHS context.

References

1. Anderson, K. M. 2001. Using structural computing to support information integration. *Proceedings of the Third Workshop on Structural Computing* (Århus, Denmark, Aug) (S. Reich et al., eds), Springer Verlag LNCS vol. 2266, 151-159.
2. Bharat, K., Chang, B.-W., Henzinger, M. and Ruhl, M. 2001. Who links to whom: mining linkage between web sites. *IEEE International Conference on Data Mining ICDM '01* (San Jose, Nov).
3. Bogelt, C. and Kruse, R. 2002. *Graphical Models: Methods for Data Analysis and Mining*, John Wiley & Sons.
4. Castillo, E., Gutierrez, J. M. and Hadi, A. S. 1997. *Expert Systems and Probabilistic Network Models*, Springer Verlag, New York.
5. Chakrabarti, S., Dom, B. and Indyk, P. 1998. Enhanced hypertext categorization using hyperlinks. *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*.
6. Chakrabarti, S., Dom, B., Gibson, D., Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan, S. and Tomkins, A. 1999. Mining the link structure of the World Wide Web. *IEEE Computer*.
7. Chen, R., Sivakumar, K. and Kargupta., H. 2001. Distributed web mining using bayesian networks from multiple data streams. *Proceedings of the 2001 IEEE International Conference on Data Mining* (San Jose, CA, Nov).
8. Dean, J. and Henzinger, M.R. 1999. Finding Related Pages in the World Wide Web. *Computer Networks 31* Amsterdam, Netherlands, 1467–1479.
9. Ghani, R., Slattery, S. and Yang, Y. 2001. Hypertext categorization using hyperlink patterns and meta data. *Proceedings of ICML-01, 18th International Conference on Machine Learning*.
10. Gibson, D., Kleinberg, J. and Raghavan, P. 1998. Inferring web communities from link topology. *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia* (Pittsburgh, PA), 225-234.
11. Haake, J. 2000. Structural computing in the collaborative work domain? *Proceedings of the Second Workshop on Structural Computing* (San Antonio, TX, May) (S. Reich, K. Anderson., eds), Springer Verlag LNCS vol. 1903. 108-119.

12. Hand, J., Mannila, H. and Smyth, P. 2001. *Principles of Data Mining*, MIT Press.
13. Hsu, J. Y-J. and Yih, W.-T. 1997. *Template-Based Information Mining from HTML Documents*, AAAI/IAAI, 256-262.
14. Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S. and Tomkins, A. S. 1999. The Web as a graph: measurements, models and methods *Lecture Notes in Computer Science, Vol. 1627.*
15. Lazarou, V. S. and Clark, K. L. 1998. Agents for hypermedia information discovery. *Lecture Notes in Computer Science, Vol. 1435.*
16. Moh, C-H., Lim, E-P. and Ng, W-K. 2000. DTD-Miner: A Tool for Mining DTD from XML Documents. *2nd IEEE Workshop on Advanced Issues of e-Commerce and Web-based Information Systems* (Milpitas, CA).
17. Ng, A. Y., Zheng, A. X. and Jordan, M. I. 2001. Stable algorithms for link analysis. *Proc. 24th Annual Intl. ACM SIGIR Conference*, ACM.
18. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As we should have thought. *Proceedings of the 1997 ACM Hypertext Conference* (Southampton, UK, Apr), ACM Press, 96-101.
19. Nürnberg, P. J., Schneider, E. R., and Leggett, J. J. 1996. Designing digital libraries for the post-literate age. *Journal of Universal Computer Science 2*(9) (Sep).
20. Pearl, A. 1989. Sun's Link Service: a protocol for open linking. *Proceedings of the 1989 ACM Conference on Hypertext* (Pittsburgh, PA, Nov), ACM Press, 137-146.
21. Punin, J., Krishnamoorthy, M., Zaki, M. J. 2001. Web usage mining: Languages and algorithms. *Studies in Classification, Data Analysis, and Knowledge Organization*, Springer-Verlag.
22. Punin, J. and Krishnamoorthy, M. 2001. Digital library portal using semantic tools in WWPal. *Semantic Web Working Symposium* (San Francisco, CA).
23. Sarukkai, R. R 2000. Link prediction and path analysis using Markov chains. *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam.
24. Segal, E., Getoor, L., Taskar B. and Koller, D. 2001. Probabilistic models of text and link structure for hypertext classification. *IJCAI Workshop on "Text Learning: Beyond Supervision"* (Seattle, WA, Aug).
25. Wiil, U., Nürnberg, P., Hicks, D. and Reich, S. 2000. A development environment for building component-based open hypermedia systems. *Proceedings of the 2000 ACM Hypertext Conference* (San Antonio, TX, May), ACM Pess.
26. Ypma, A. and Heskes, T. 2002. Categorization of web pages and user clustering with mixture of hidden markov models. *WEBKDD 2002*, (Canada).
27. XML Linking Language (XLink) Version 1.0. `http://www.w3.org/TR/xlink/`

EMMOs – Enhanced Multimedia Meta Objects

Sunil Goyal, Siegfried Reich, and Wernher Behrendt

Salzburg Research/SunTREC
Jakob Haringer Straße 5/III
5020 Salzburg, Austria
{sgoyal, sreich, wbehrendt}@salzburgresearch.at

Abstract. There exist a broad variety of concepts and tools for supporting knowledge workers during the various phases of their work. However, structures created with one tool — e.g. a mindmap editor capturing a discussion process — usually cannot be re-used by other tools or in other contexts. The reasons are manifold, ranging from variations in syntactic expressions to semantic heterogeneity. In this paper we argue that there is a need for capturing and representing explicitly the structures not only between multimedia objects but also between the various stages in a development process. We outline a document model which we envisage to store these relationships and we name that model "EMMO — Enhanced Multimedia Meta Object". EMMOs are enhanced by explicit knowledge structures, and they are first-class, i.e., they can be exchanged, traded and re-used in different contexts.

1 Introduction

Knowledge workers require support for re-purposing of knowledge assets composed of multimedia documents. This paper addresses the domain of interactive electronic publishing. Our rationale is that the "document of the future" will be generated on the fly, based on knowledge assets, will contain any type of media, and will be available in appropriate form, for interactive use, to humans as well as machines, anywhere, when needed.

These visions date back to the early ideas of the hypertext pioneers, including Bush [2] or Engelbart [4]. Today, there exist a large number of tools for the different application domains [8]: browsers for the various breeds of navigational hypertext, spatial browsers for the domain of spatial hypertext [6], and editors and viewers for issue based information systems [3], for example QuestMap^{TM} [9]. Exchange between these and re-purposing of existing relationships in different contexts, however, is still an open issue. We argue that there is a need for an inclusive model that manages the relationships between the different models.

Let us consider the following example.

Two years ago, Mr. Smith, CEO of Sporty-Cars, Inc. had put forward his vision – a Next Generation Sports Car – using MS PowerPoint^{TM} slides for his upcoming board meeting. The market research done earlier had shown impressive demand. Everyone at the board meeting cheered. The product design teams

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 155–160, 2003.

went through a rigorous argumentation process and captured it using tools like QuestMap^{TM}. They discussed every single issue, prepared manuals, built CAD simulations and 3D models. Everything looked impressive and the project was given a green light for prototype development. It was a great day for the company. Mr. Smith along with his colleagues came to see the prototype. Mr. Smith got into the new car and was suddenly plunged into darkness. The car seat had sunk down! Was he too heavy for the seat?

Mr. Smith wants to know where exactly did things go wrong? Which events led to what? But where should he look and analyze? In the MS PowerPoint^{TM} slides, QuestMap^{TM}, manuals, CAD simulations or 3D models?

Current tools often lack explicit semantics, they pose interoperability problems due to semantic heterogeneity and they are therefore unable to solve the above problem.

2 EMMOs — Enhanced Multimedia Meta Objects

EMMOs provide a concept of portable, stand-alone and living meta objects which assist users in traversing associations between artefacts. EMMOs are objects that describe complex relationships between artefacts, can be loaded into browsers and modified by users. They also keep the history of interactions between the resources and their users and set individualised viewpoints over the information space. Therefore, EMMOs cover the development process and allow for knowledge assets to be re-used in different contexts. Figure 1 gives a graphical representation of an EMMO. In particular, the figure shows the different interfaces for accessing and manipulating the internal structures.

An EMMO is composed of the following structure:

- *A knowledge template* holds domain intelligence pertaining to the universe of discourse.
- *A set of transformation descriptions* to allow the EMMO knowledge to be represented in various output formats.
- *A links repository* holds instantations of relationships between artefacts in digital libraries. The available types of relationships are specified by the knowledge template (see above).
- *A contents repository* represents a persistent store which holds additional information – e.g. annotations pertaining to the instance.
- *A trails repository* keeps history of the usage of the EMMOs link repository.
- *A set of standard interfaces* for authoring, an inspection environment, trading interface, resource (self) description interface, module management and import/export interfaces.

The EMMO concept came originally from the domain of intertextual studies in literature and arts where scientists want to make explicit the different relationships between texts in a way that approximates the process of cultural contextualization characteristic of interpretative processes (see also the project CULTOS — Cultural Units of Learning, Tools and Services `http://www.cultos.org/`).

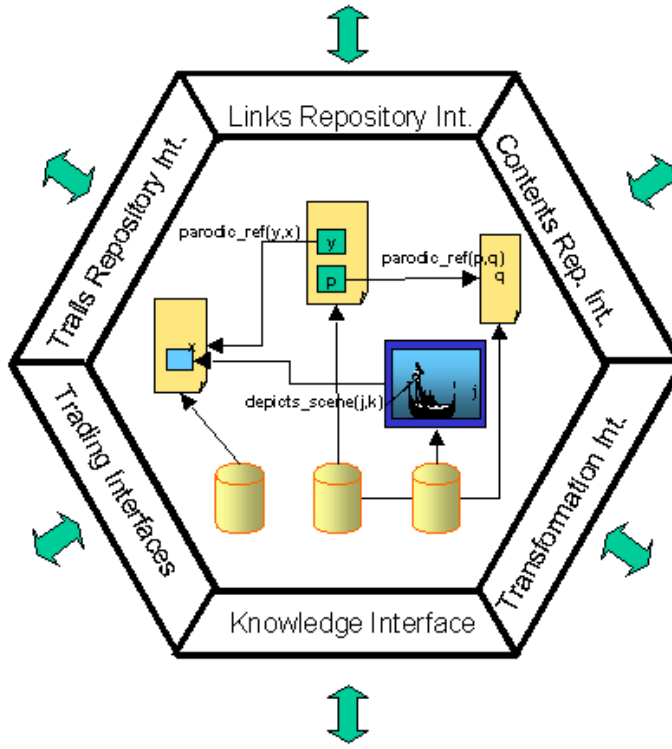


Fig. 1. EMMO structure

Within our work we produce an authoring environment for intertextual threads (which are based on EMMOs).

An emmo representing cultural concepts and associations between various cultural artefacts can be re-purposed and reused by cultural scientists, students or teachers in different perspectives. A cultural scientist can be interested in certain relationships which might not hold significance for a novice user. Knowledge template, links repository help to define and capture the complex relationships that exist within the cultural domain. Annotations and trail interfaces assist a particular user in holding rich discussions amongst his peers. A set of transformation descriptions assist the user to view the content in different formats - a rich multimedia presentation in SMIL or a handy ebook in PDF or view it online in HTML via World Wide Web. Standard interfaces like Import/Export help in easier exchange of Emmos a mongst peers. All these above interfaces form an integral part of the Emmo there by enabling Emmos to be able to capture not only rich semantics but also providing the ability to re-purpose and reuse these captured semantics in different contexts.

Figure 2 shows an example of an intertextual thread, “The Fall”, a poem by Tuvia Rivner. In particular, it shows the relationships that exist between

various cultural texts within this thread. The numbers in the Figure can be interpreted as follows: the title "The Fall" is associated with several cultural concepts, e.g., the "snake", "forbidden fruit" or even "accusation of the woman". Secondly, a reference from a text to an image is referred to as an "ekphrasis". In the particular poems there are two of these references, one to a painting of Chagall, another to a painting of Breughel. Thirdly, descriptive configurations allow other works (in this case Milton's "Paradise Lost") to be referred to. The set of possible relationships is being controlled by an underlying ontology. The ontology is domain specific, it assists for different perspectives and it is developed with general consensus within a community.

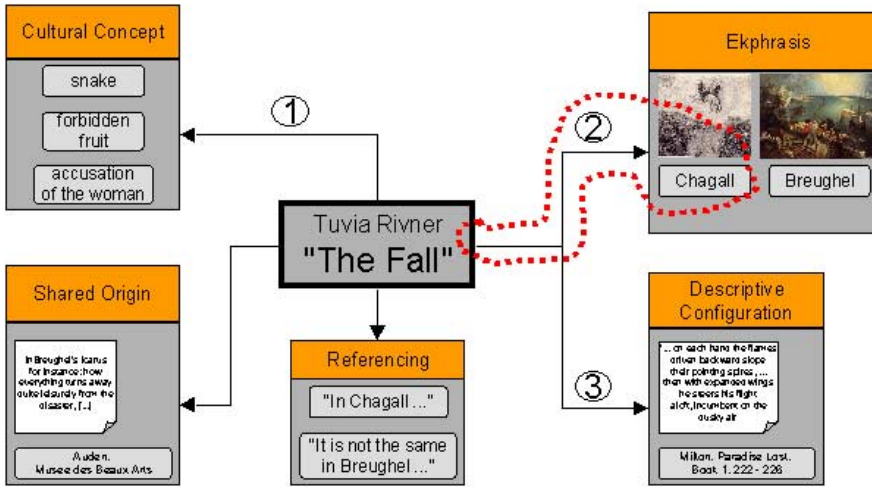


Fig. 2. An Intertextual Thread: "The Fall"

One of the major aims of the CULTOS is to deliver and deploy these portable meta-data objects, implemented as EMMOs, in different publishing scenarios. We believe that the complex relationships that exist within knowledge assets can be captured using EMMOs so that they can be reused in a different context and can be applied to a variety of different domains:

- *Cultural Heritage*: "soft" semantics (different viewpoints, fuzzy separations, etc.)
- *News Publishing*: generating types of news on the fly from archives.
- *Education*: support in the form of trail based systems, argumentation support, community building tools etc.
- *Engineering discipline*: vast amount of technical documentation combined with questions such as "why did we design that?".
- *Executive Knowledge Management*: support of knowledge management and decision support systems.

The idea of autonomous knowledge objects has been proposed in several variations. *Tsichritzis* [11] envisioned mobile knowledge objects (KNOs) to be used in software engineering and office automation. *Memes* [1] followed the approach that behaviours and ideas are copied from person to person by imitation and *Tanaka* [10] proposed a framework for building, managing and trading Memes. However, we argue that these approaches are not based on an underlying ontology, they are often domain specific and they do not address issues of semantic heterogeneity; therefore, they are unable to capture and support the whole development process.

3 Summary and Discussion

We have outlined the idea of multimedia meta objects which are enhanced by knowledge and which capture the relationships between knowledge assets so that they can be re-used in different contexts such as news publishing, technical documentation, automotive engineering or software development etc.

Hence, we believe the EMMOs concept to be general and applicable to multiple domains (which is what would make it a "C-level" concept following Engelbart's categorisation, see `http://www.bootstrap.org/`).

Discourse, trails, perspectives, knowledge extraction, and others, are key concepts along with knowledge assets which provide assistance to knowledge workers during various phases of their work. We believe that there is a need for an infrastructure (C-level) to support the above needs of knowledge workers and applications can leverage this infrastructure and re-purpose it according to their own needs. More specifically, we believe that the following components are part of that infrastructure:

- a structure server supporting multiple interaction styles or hypermedia application domains [7]);
- an ontology system that manages relationships based on ontologies (which in turn capture a domain's or community's concepts);
- a process support system that allows users to keep track of the status of their work [5]. This could well be an adapted workflow management system;
- a file system which supports users' so that they can manipulate EMMOs with their existing tools;
- viewers and transformers allowing for different representations of knowledge assets.

Acknowledgements

CULTOS is co-funded by the European Commission under the IST Programme (Information Society Technologies) under grant No. IST-2000-28134.

References

1. Blackmore, S. 2000. The power of MEMES. *Scientific American 280* (Oct), 52-61.
2. Bush, V. 1945. As we may think. *The Atlantic Monthly 176*(1), 101-108.
3. Conklin, J. H. C. and Begeman, M. L. 1988. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems 6*(4), 303-331.
4. Engelbart, D. C. 1988. *Computer-Supported Cooperative Work: A Book of Readings* (I. Greif, Ed.), chapter Toward High-Performance Knowledge Workers, Morgan Kaufmann.
5. Haake, J. M. and Wang, W. 1999. Flexible support for business processes: Extending cooperative hypermedia with process support. *Information and Software Technology 41*(6), 355-366.
6. Marshall, C. C. and Shipman, F. M. 1997. Spatial hypertext and the practice of information triage. *Proceedings of the '97 ACM Conference on Hypertext* (Southampton, UK, Apr), 124-133.
7. Nürnberg, P. J., Grønbæk, K., Bucka-Lassen, D., Pedersen, C. A., and Reinert, O. 1999. A component-based open hypermedia approach to integrating structure services. *New Review of Hypermedia and Multimedia*, 179-205.
8. Nürnberg, P. J., Leggett, J. J. and Schneider, E. R. 1997. As we should have thought. *Proceedings of the '97 ACM Conference on Hypertext* (Southampton, UK, Apr), 96-101.
9. Shum, S. B. 1998. Negotiating the construction of organisational memories. *Information Technology for Knowledge Management* (U. M. Borghoff and R. Pareschi, Ed.), 55-78.
10. Tanaka, Y. and Fujima, J. 2000. Meme media and topica architectures for editing, distributing and managing intellectual assets. *2000 Kyoto International Conference on Digital Libraries: Research and Practice* (Kyoto, Japan, Nov), 208-216.
11. Tsichritzis, D. C. 1985. *Office Automation, Concepts and Tools*, chapter 15 Objectworld, Springer, 379-398.

A UML Profile for Agent-Based Development

Amir Zeid

Department of Computer Science
The American University in Cairo
113 Kasr El Aini St
Cairo, Egypt
azeid@aucegypt.edu

Abstract. In recent years, agent-based systems have received considerable attention in both academia and industry. The agent oriented paradigm can be considered a natural extension to the object oriented (OO) paradigm. In spite of the fact that there are many OO analysis and design methods, there is very little work reported on design and analysis of agent-based systems. Agents differ from objects in many issues which requires special modeling elements. In this paper, we propose a software engineering process for agent-based systems based on existing object-oriented software engineering concepts. We extend the Unified Modeling Language (UML) to model agents (static and mobile).

1 Introduction

Agent-Oriented (AO) technology builds upon and extends the now widely accepted Object-oriented (OO) approach to system design. The OO paradigm views an application as being composed of entities called objects that interact with each other by sending and receiving messages. Objects are grouped into classes. Then, class relationships can be captured in separate hierarchies. Objects are reactive entities as they receive messages and react to specific external events. The AO paradigm goes one step further by viewing an application as being made up of one or more autonomous agents. Each agent continuously receives input. Agents respond to the environment where they reside by taking actions that affect the environment. Agents are pro-active entities, evaluating situations and then acting accordingly. Agents do not require specific instruction messages. Current applications being developed based on AO technology include Spacecraft Fault Diagnosis, Management and Business Process Management, Air Traffic Management, Air Mission Simulation and Telecommunication Network Management. Objects in Object-Oriented Programming (OOP) represent real-world counterparts such as inventory items, documents, financial transactions and business processes. Objects are an intuitive mapping of the problem space to a computational model. They are characterized by state and defined behavior. They interact by passing messages since object internals are obscure. Agent-oriented programming deals with both object and agent primitives. In some ways, agents may be thought of as objects, because they are also characterized by state and behavior. Unlike objects, agents have goals and proactively

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 161–170, 2003.

work to achieve them. On one hand, object methods are simply procedures coded by a programmer to deal with different messages. For example, a dog object can bark when pinched. On the other hand, an agent can synthesize plans to deal with contingencies not foreseen by the programmer and can learn from experience. For example, a dog agent can discover that insistent barking gets him fed a lot quicker. The rest of this paper is organized as follows:

- Section 2 gives a background about agent-based software engineering and the UML (Unified Modeling Language).
- Section 3 describes our proposed process and extensions to the UML.
- Section 4 concludes our paper and suggests research directions.

2 Background

In this section, we will discuss the differences between agents and objects. A brief description of the UML and one of its supporting tools (Rational) will also be introduced.

2.1 Agents vs. Objects

After a thorough investigation of OO analysis and design techniques it was found that they are not directly applicable to the development of multi-agent systems. This is basically due to the conceptual differences between objects and agents:

- Agents have a more complex behavior and structure than objects, and in this respect they are more comparable to subsystems in some OO methodologies, (e.g., [11]). Their internal structure differs from objects since agents have a more complex underlying functional architecture such as the belief-desire-intention (BDI) architecture [10]. In this respect they are on a higher level of abstraction than objects.
- In contrast to objects that are rather passive entities, agents are active. On one hand, objects immediately become active through messages and in this respect they are benevolent. On the other hand, agents act on their own behalf by following their goals, and can decide whether they act and respond to the events and messages received from other agents.
- Another difference between objects and agents is that objects usually do not have any autonomy as opposed to agents, which are characterized by their aims. Each agent follows its aims autonomously and independently, without being given instructions from outside.
- Mobility modeling is not handled by existing OO methods.

2.2 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for

business modeling and other non-software systems. The UML represents a collection of engineering practices that have proven successful in the modeling of large and complex systems. The UML is a language, it is not just a notation for drawing diagrams. The UML is not a process, it provides a modeling language without enforcing a lifecycle. The UML can be extended to model new elements using the concepts of stereotyping and tagged values. Stereotypes are a mechanism for introducing new types of modeling elements. Stereotypes are strings enclosed in <<>>, preceding the name of an element. They may have an associated graphic icon or graphic marker. The new modeling elements that can be introduced using stereotypes must be subclasses of existing modeling elements [1].

2.3 Rational Suite

The Rational Suite package is an integrated, full-lifecycle solution for software architects, designers, analysts, and developers. Available in Windows, UNIX, and RealTime editions, Rational Suite gives the power to create more resilient, component-based architectures; automatically generate test harnesses for thorough model- and scenario-based testing; and utilize run-time tools that automatically pinpoint hard-to-find bugs, highlight performance bottlenecks, and find untested code [9].

2.4 Related Work

Agent-based analysis and design methods can be categorized into two main groups:

- Methodologies based on object-oriented methodologies: [3,2,7].
- Methodologies that adapt knowledge engineering: [5,8].

Our proposed profile belongs to the first category since it is based on UML concepts.

3 Proposed UML Profile

As mentioned before, we propose a process based on UML to model agents (static and mobile). The process covers both analysis and design phases.

3.1 Analysis

Analysis aims to clarify what a system is supposed to do, it mainly aims at answering "what" the system does rather than "how" to do it. The proposed analysis process is illustrated in Fig. 1. The first step is to define the expected tasks of the system. The second step is to identify "places", agents and classes. Then, a static model of the identified agents, classes and places is constructed. The analysis process is concluded by filling a data dictionary.

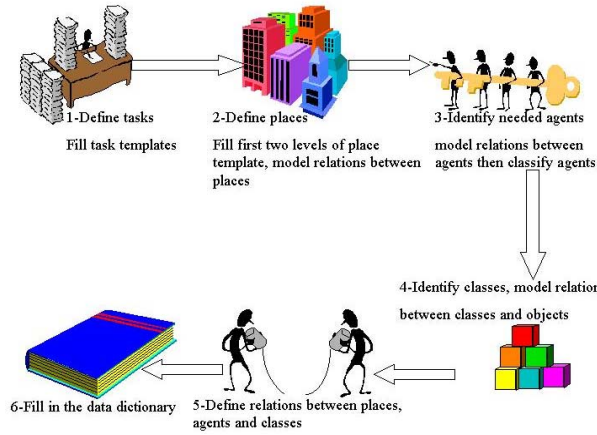


Fig. 1. The Analysis Process

Identify Tasks. Agent-oriented paradigm is goal-oriented. Consequently, the first step should be defining expected tasks because agents will be identified based on these tasks. Figure 2 gives the syntax of the task template. A task template will be filled for each task. The template includes the task name, its super-task (if it has any), sub-tasks (if it can be sub-divided), alternatives to achieve the task and exceptions.

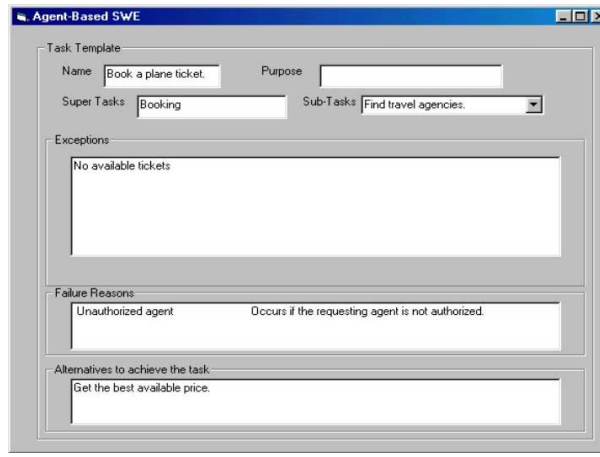


Fig. 2. The Task Template

Identify Places. The second step is to define "places". The concept of places is associated with mobile agents. When a mobile agent transfers itself, it travels between execution environments called places. A place is a context within an agent system in which an agent can execute. This context can provide functions such as access control. The source place and the destination place can reside on the same agent system, or on different agent systems that support the same agent profile. A place is associated with a location, which consists of the place name and the address of the agent system within which the place resides. When a client requests the location of an agent, it receives the address of the place where the agent is executing. A place template will be filled for each identified place. The template contains place name, purpose and super-place (if it has any). Figure 3 shows the syntax of a place template. The following level describes the communications among places, which places are safe to accept agents from, which places are authorized to send agents to. The last level of the template contains the attributes and methods of the place.

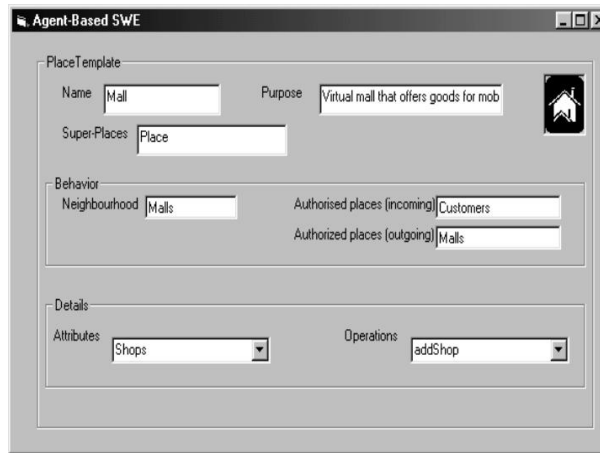


Fig. 3. The Place Template

Identify Agents. Agents can be considered similar to active objects in UML. In order to be able to have secure environments for mobile agents, the concept of agent's owner is utilized. The owner of an agent can be defined as the place where it was created or originated from. The owner can later be used as a security identifier for the mobile agent. As for mobile agents there are three main actions that they can do. Mobile agents can go to a place, meet another agent who resides in the same place and connect to another agent on another place. A good starting point to identify candidate agents is to assign an agent for each task identified earlier in the first step. Moreover, agents can be identified for other purposes such as representing users, devices and databases. Figure 4 shows an example of

agent template. The next step is to classify the identified agents. First, classify them in terms of mobility, whether they are static or mobile. Then agents will be classified according to their functionality:

- Interface agents are agents that interact with the user receiving user specifications and delivering results. They acquire, model and utilize user preferences to guide system coordination in support of the user's tasks. The main functions of an interface agent include: collecting relevant information from the user to initiate a task, and asking the user for additional information during problem solving.
- Task agents are agents that support decision making by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. A task agent receives user delegated task specifications from an interface agent and forms plans to satisfy these goals. Task agents are then classified into three categories:
  - Watcher agents are agents that continuously monitor a stream of information and notifies its customer when it finds out;
  - Searcher agents are agents that actively search;
  - Analyst agents are agents that make conclusions out of given data.
- Information agents are agents that answer a one-shot query about associated information sources.

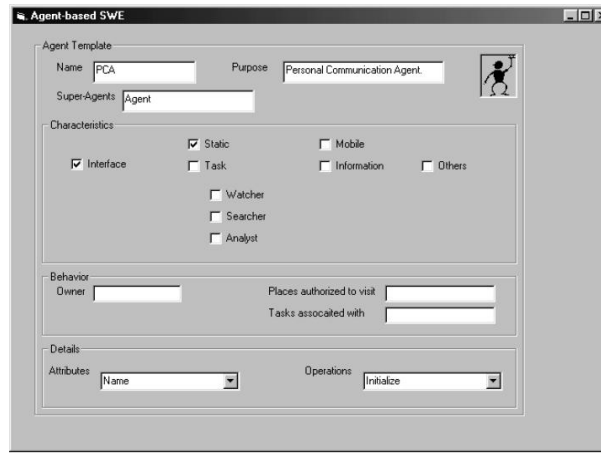


Fig. 4. Agents Template

Identify Classes and Objects. This step can be done using any of the existing OO methods. We chose Unified Modeling Language (UML) notations since UML is the most commonly used set of notations.

Model Relationships between Agents, Classes and Places. In this step, a static view of the whole system is constructed. The aim is to describe the relationships between agents, classes and places in one global view. Figure 5 illustrates the notations that will be used. It includes the notations for static and mobile agents, classes and places.

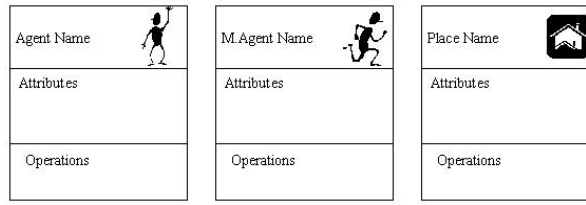


Fig. 5. Static and Mobile Agents Icons

Create Data Dictionary. The concept of a data dictionary has been used in traditional software engineering and database tools for many years. A data dictionary is a central repository of definitions of terms and concepts. Without it, the process has little semantic content. A data dictionary is a vital tool by virtue of its role as a single place to look up definitions. It provides clarity of thought and transition from phase to phase. A further important reason for having a data dictionary is that it helps someone unfamiliar with the development to understand it. Without a data dictionary the unfamiliar reader is often forced to rely on decoding meaningful identifiers. A set of shared vocabulary and meaning for concepts (often called an ontology) is then defined to conclude the analysis process.

3.2 Design

Design deals mainly with how to implement the system to produce the expected outcomes as stated in the analysis phase. The proposed design process is illustrated in Fig. 6. Design starts by defining authorization rights for places. After that, state machines for agents, places and classes are constructed. Then, a life-cycle model (Sect 3.2) is constructed for each major task. Related places are then grouped in neighborhoods. The next step is to describe the dynamic behavior of the system by defining mobile agents itineraries and modified interaction diagrams (based on Jacobson's software engineering process [6]). The design phase is concluded by detailed design for methods (operations) and attributes of classes, agents, and places. Finally, the data dictionary is updated.

Define Authorization Rights. The concept of authorization rights for each defined place is introduced because security issues have to be taken into consideration when dealing with mobile agents. Authorization rights indicate which

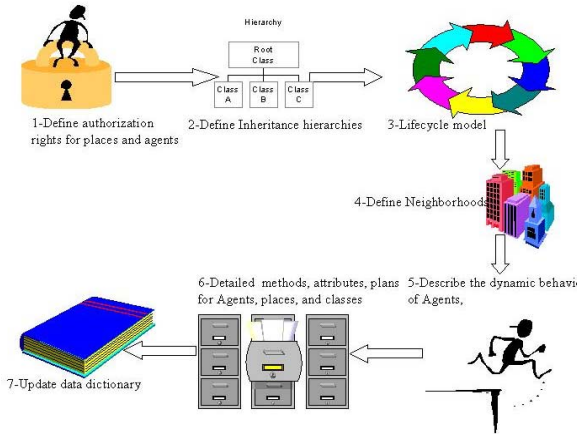


Fig. 6. The Design Process

agents are allowed to visit a certain place (their owner must be trustworthy). Places templates will be updated accordingly.

Define State Charts for Agents, Classes and Places. State diagrams render the states and responses of a component. These diagrams describe the behavior of a class in response to external actors. These diagrams contain states and transitions. We use the UML notations for state diagrams.

Life-Cycle Model. The life-cycle model describes the behavior of agents and how they communicate with the environment from creation to death. A life-cycle expression defines the allowable sequences of interactions that an agent may participate in, over its lifetime. The Rules of life-cycle model are as follows:

- Alphabet: Any input or output event maybe used in an expression.
- Operators: Let x and y be life-cycle expressions, then:
- x.y denotes x is followed by y.
- x|y denotes either x or y occurs.
- X* denotes zero or more occurrences of x.
- X+ denotes one or more occurrences of x.
- X||y means arbitrarily interleaving the elements of x and y.
- [x] denotes that x is optional.

The idea is applied in the Fusion method [4]. Life-cycle expressions define patterns of communication. As an example consider a simple banking system. A very simple pattern occurs in response to balance inquiries. A check-balance event or task is always followed by the output of a current balance event. An example would be Inquiry= check-balance. current-balance.

Define Subsystems and Neighborhoods. The next step is to start defining subsystems and neighborhoods. Subsystems are groups of agents, classes and places that collaborate together to perform a set of related tasks. First, we have to introduce the concept of a "Neighborhood". Two places are in the same "neighborhood" if they are physically on the same host; i.e. agents do not have to travel from one place to another to perform a task if the two places are in the same neighborhood. Neighborhoods are useful to keep track of mobile agents. The main goal is to minimize agents roaming over the network.

Describe the Dynamic Behavior of the System. An itinerary graph is constructed for each scenario a mobile agent participates in. The itinerary graph shows places where the mobile agent passes by to achieve a certain task. Figure 7 shows an example of the itinerary graph of a personal travel agent. The example shows a mobile agent (Personal Travel Agent) that travels between 3 places (Personal PC, Flight Status Place and Paging Place) in order to book a ticket for a customer. The character indicates where the mobile agent started its itinerary. The number on each place indicates the number of the detailed interaction diagram that describes the interactions between the mobile agent and agents/classes in that place. For each identified task, an interaction diagram will be constructed.

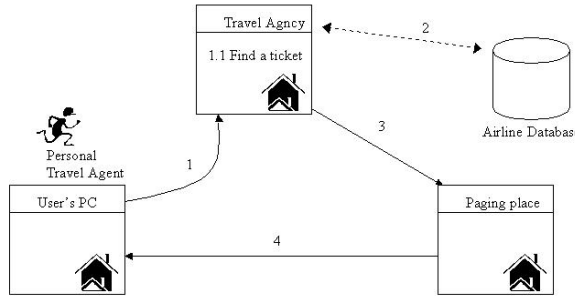


Fig. 7. Itinerary Graph

3.3 Stereotypes (Extensions to UML)

In this section, we include a list of the stereotypes we added to the UML. Stereotyping is the formal method to extend UML as mentioned before in Sect. 2.2.

1. <<Agent>> Class
2. <<Mobile Agent>> Agent
3. <<Place>> Class
4. <<Neighborhood>> Package

5. <<Agent Template>> Package
6. <<Itinerary graphs>> Package
7. <<Place Template>> Package
8. <<Life Cycle>> Package
9. <<Task Template>> Package
10. <<Interface Agent>> Agent
11. <<Task agent>> Agent
12. <<Info. Agent>> Agent
13. <<go>> Operation
14. <<meet>> Operation
15. <<dispose>> Operation
16. <<connect>> Operation

4 Conclusion

In this paper we presented a new UML profile to model agents (static and mobile). The new profile includes notations for some new elements like agents, places and neighborhoods. The profile includes some new diagrams like the itinerary graph, authorization graph. The profile also includes some modifications to existing diagrams like the interaction diagram. Future work will include code generation for agent-based languages.

References

1. Alhir, S. 1998. *UML in a Nutshell*, O'Rielly.
2. Bauer, B, et al. 2000. Agent UML: a formalism on specifying multi-agent software systems. *Proceedings of the First International Workshop (AOSE-2000).*
3. Burmeister, B. 1996. Models and methodologies for agent-oriented analysis and design. *DFKI document D-96-06.*
4. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. 1994. *Object-Oriented Development. The Fusion Method*, Prentice Hall, Englewood Cliffs.
5. Iglesias, C. et al. 1998. Analysis and Design of multi-agent systems using MAS-commonKADS. *Intelligent Agents, IV, LNAI V 1365*, Springer, Berlin.
6. Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. 1992. *Object-Oriented Software Engineering. A Use Case Driven Approach*, ACM Press/Addison-Wesley.
7. Kinny, D. 1993. "The Distributed Multi-Agent Reasoning System Architecture and Language Specification", Australian Artificial Intelligence Institute, Melbourne.
8. Luck, M et al. 1997. From Agent theory to agent construction: a case study. *LNAI V 1193*, Springer, Berlin.
9. `http://www.rational.com/uml/`
10. Rao, A. S., Georgeff, M. P. 1992. "An Abstract Architecture for RationalAgents" (Nebel, B., Rich, C., Swartout, W., Eds.) *Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, Morgan Kaufmann, San Mateo.
11. Wirfs-Brock, R., Wilkerson, B., Wiener, L. 1990. *Designing Object-Oriented Software*, Prentice-Hall, Englewood Cliffs.

Widening the Configuration Management Perspective

Lars Bendix

Department of Computer Science
Lund Institute of Technology
P. O. Box 118, S-221 00 Lund, Sweden
bendix@cs.lth.se

Abstract. A metainformatics symposium provides an excellent opportunity to explore, discuss and investigate how configuration management techniques and principles can be expanded into new fields and how it itself can incorporate techniques from other fields.

1 Software Configuration Management

For a long time Software Configuration Management (SCM) has been considered an important part of software engineering and something that is indispensable to carry out on software development projects. SCM is the discipline of organising, controlling and managing the development and evolution of software systems. In particular, emphasis is put on: configuration identification to uniquely identify a product and its parts; configuration control to apply changes to a product in a disciplined and orderly way; configuration status accounting to collect and report data about the status of the development of the product; and configuration audit to verify and validate that the right product has been built and built the right way. Thus in general SCM is seen as a vehicle to ensure consistency and quality of the product being produced.

This is, however, just one face of SCM, as something that satisfies the needs of a company that has to deliver products to customers. Another face of SCM is that presented by Babich [2] in 1986, where he considers SCM as a means for obtaining team productivity by way of co-ordinating the efforts of a group of people. In his opinion the facilitation of collaboration and synchronisation of work should be a main focus of SCM. Thus focus is shifted – or widened – from consistency and quality of the product itself to consistency and quality in the workflow processes of the people that have to create the product. Unfortunately the traditional SCM world has not been very responsive to these needs so far.

At Lund University we have looked for some time at how developers can use SCM in their process of creating a common product. We have analysed how SCM is used in some Open Source Software projects [1]. It turned out that the traditional concepts of SCM were not used – at least explicitly – and that what was actually used were techniques for co-ordination of work, simple version control, build management, selection of configurations and handling of

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 171–176, 2003.

workspaces. Currently we are analysing how 12 groups of students – 8-10 people in each group – used SCM in projects where they developed an application following the Extreme Programming methodology. They used CVS as the tool for SCM and asked whether it had been useful, most of them said that it would not have been possible to carry out the project without it. Preliminary results, however, show that in most cases CVS was used only for simple synchronisation and integration of individual work.

In general, it is not just developers that collaborate to create a common product and as such can benefit from SCM techniques. In CSCW, people have been looking at how to facilitate collaborative writing and their results have many things in common with traditional SCM mechanisms for version control. The hypertext community has also dealt with aspects of versioning and especially problems regarding referential integrity, which is also a concern in SCM. In most software development environments, SCM is an important tool, which means that aspects such as information retrieval and visualisation have to be dealt with. Furthermore, aspects of programming languages also has impact on the build management part of SCM.

SCM can be considered the foundation for any kind of collaborative – or individual for that sake – effort and as such, it has relations to many other fields. This means that SCM needs input from these fields to typecast its general solutions so they can be tailored for each specific context.

2 Conquering New Territory

The above mentioned aspects of collaboration and co-ordination is one important face of SCM. It has been sadly neglected by researchers and industry and because of that practitioners are struggling. Another face of SCM, that has had the interest of researchers and industry for years, is Product Data Management (PDM). It deals with the management and control of data about a product and recording changes to these. PDM is a mature discipline with well established techniques and principles and as such has the potential to be applied to other domains to conquer new territory.

The motive for collecting these data is to keep a trace of what has happened to a product, in case we need to investigate the cause for a certain problem. In that respect it is quite similar to the flight recorders used on board aircrafts. In the case of a software disaster we just go and look for the black box – which in real life is orange – and pull out all information necessary to analyse and establish the cause of the disaster. In its simplest case, it is plain version control and just records one parameter: what changed (and implicitly that something was changed). In a full-fledged SCM solution, it would record a whole range of parameters such as: when, why and by whom a change was made, the associated change request and/or bug report, related changes, and more.

Knowledge management is one domain of potential expansion for SCM systems. Rubart et al. [9] points out that knowledge management is co-operative. So are SCM systems and they can be used to create, disseminate and utilise

any kind of knowledge. Most SCM systems also have built in processes for the creation, dissemination and utilisation of knowledge and in many cases these processes are tailorable. The actual implementation of the SCM system might have to be changed to adapt to the needs of new domains like knowledge management, but the techniques and principles are ready and mature.

Tochtermann [10] states that knowledge management software is primarily used to mediate between people and to provide team work spaces – that is exactly what SCM tools can do as explained in Sect. 1. Tochtermann claims that as a consequence knowledge management tool developers should know what knowledge is. In their particular domain, SCM tool developers is a striking example of that, as in most cases they use the SCM tool to develop the SCM tool. This means that SCM is actually – as Tochtermann states knowledge management should be – driven by use needs and not technology, and therefore "generate the highest benefit for the users with the slightest effort".

According to Dalcher [4], modern software development should be seen as a very change-intensive activity. Emphasis should be shifted from the product to the processes used to produce the product, and a life-time perspective should be applied to include also maintenance and further evolution. Even such dynamic scenarios can be supported by modern SCM systems. They provide for the continuity dimension that spans from early development people to the later arrival of maintenance people. Acting as a common repository for all parties with interest in the development phases, an SCM system allows a continuous, as opposed to discrete, view of the development effort. Such a shared, common repository can conserve the intellectual capital that Dalcher says will become a primary resource for software development companies.

3 Adopting New Techniques and Tricks

SCM is an expanding field and moving into other domains, it needs to incorporate new ideas from these domains. And even when it is not expanding, SCM needs to consolidate its own practices and thus can use inspiration from other fields. SCM as a discipline has been around for only about three decades. However, CM – without the software – has existed for ages and when it moved into SCM also had to adapt to the characteristics of the new domain. One of the earliest applications of CM principles were probably when God put together man, discovered that He was not quite satisfied with that and went on to make some changes to create Woman. Using CM He would be able to roll back the product if the new version turned out to be worse. Fortunately He still has not undone his latest creation and reverted to the previous version. So SCM – and in particular CM – is a mature discipline. Despite that there are still some problems around in SCM that could do with better solutions. Two such problems are: conceptual models for versions of configurations and selection of configurations.

It is possible to provide the user – whether a software developer or someone else – with a conceptual model for versioning of an object that is simple. The same is true for the conceptual model for a configuration, but when we

start to put versions and configurations together things start to get complex. A conceptual model for the fact that there may exist several versions of a particular configuration is not simple at all – and even worse when we want to model the possible configurations that can be created from the versions of a given set of objects. It is possible to model these cases by the use of and/or-graphs [11], but they become very complex. Another, more serious, problem with the and/or-graphs is that they do not have any abstraction capabilities, such that we always get the full picture. The containment models of Gordon and Whitehead [5], on the other hand, are able to abstract away details that we may consider irrelevant at any time. Furthermore, containment models seem to convey important SCM information in a very intuitive way. So they might be used to give better conceptual models for some fundamental SCM concepts. However, it seem that they too suffer from some of the same limitations that the and/or-graphs have: the inability to model dynamic dependencies between objects and to model the fact that a versioned object might be split into two at some point. This could be caused by the restrictions that are placed on the specialised entity-relationship model and might be solved by relaxing the restrictions to obtain a richer, though still simple, model. It would be interesting future work to incorporate relations between atomic objects and containers to model dynamic dependencies, and to introduce a new "split-from" semantics for relations to model versioning of split objects.

During the co-operative work of a group, new versions of objects are endlessly added to the repository. At points in time, a developer may want to limit his view of the repository and create a context in which he can work or a configuration he can compile. In traditional SCM, this is done by using a generic configuration as description and perform a selection from the repository based on the description and a number of attribute values. What is obtained is a fully- or partial-bound configuration. This model is simple and has the advantage that it allows selections to be performed in parallel. However, it hardly reflects the needs of modern evolutionary software development methods. When the software evolves, the architecture (the structure) will also change and generic configurations are not able to model dynamic structures being static by nature. To support such development methods, SCM must provide for versioning and selection on structures. Griffiths et al [6] provide a solution for that in a hypermedia setting. Each object and relation comes with attached to it a context that can then be used during the query process to decide the visibility of data. More importantly, they change the selection process such that it is no longer performed statically and globally, but locally step-by-step and varying dynamically based on the results from the previous steps. It should be possible to adapt such a process of pattern propagation to the software development domain to obtain a richer and more flexible selection mechanism than current SCM systems offer.

4 Reflections

A metainformatics symposium has shown to be an exciting and fertile breeding ground for interdisciplinary exchange of ideas. In this paper, we have pointed out some domains where it should be possible to use SCM techniques and principles in the modest hope that people in those domains will find inspiration. Furthermore, we have seen some interesting and promising ideas that it should be possible to adopt and incorporate into SCM to make it stronger.

Hicks [7] describes the A, B and C levels of work and ask the question: where are the B-users to use the C-level open hypermedia systems? They are probably home working on implementing the A-level tools directly! To cast the levels in a different domain, then at the A-level you use compilers, at the B-level you construct compilers using the compiler generators provided by the C-level people. Now the question can be rephrased: why do I hand-build a compiler instead of using a compiler generator? There could be many reasons, I might need the compiler to be optimised for speed and storage or it might simply be that it is easier and faster for me to handcraft a one-off compiler to convert from one format to another than it is to work out how the compiler-generator gets started. Probably these C-level products will always either have all the bells and whistles – and be unusable bloatware – or be slim and to the point – forcing me to implement or tailor all the things that I really do need. Maybe we would be better off if we considered it a success when we were able to build metainformatical bridges by sharing best practices and reusing processes and principles between domains [8], instead of trying to sell our C-level tools. Maybe they just serve to play around with to get hold of the right principles.

References

1. Asklund, U. and Bendix, L. 2002. A study of configuration management in open source software. *IEE Proceedings–Software 49*(1) (Feb).
2. Babich, W. A. 1986. *Software Configuration Management - Coordination for Team Productivity*, Addison–Wesley.
3. Bendix, L. and Vinter, O. 2001. Configuration Management from a Developer's Perspective. *Proceedings of the EuroSTAR 2001 Conference* (Stockholm, Sweden, Nov).
4. Dalcher, D. 2003. Software development for dynamic systems. (this volume).
5. Gordon, D. and Whitehead, Jr, E. J. 2003. Containment modeling of content management systems. (this volume).
6. Griffiths, J., Millard, D. E., Davis, H., Michaelides, D. T. and Weal, M. J. 2003. Reconciling versioning and context on hypermedia structure servers. (this volume).
7. Hicks, D. 2003. In search of a user base: Where are the B's? (this volume).
8. Nürnberg, P. J. 2003. Building metainformatical bridges. (this volume).
9. Rubart, J., Wang, W. and Haake, J. M.. 2003. A meta-modeling environment for cooperative knowledge management. (this volume).

10. Tochtermann, K. 2003. On the role of computer science in knowledge management. (this volume).
11. Tichy, W. F. 1982. A data model for programming support environments and its application. *Automated Tools for Information Systems Design*, (Schneider, H. J. & Wasserman, A. L., Eds.), North-Holland.

Web Services in ERP Solutions: A Managerial Perspective

Bostjan Kezmah and Ivan Rozman

Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ulica 17
SI-2000 Maribor, Slovenia
{bostjan.kezmah,ivan.rozman}@uni-mb.si

Abstract. The latest promising technology called web services carries a potential to expand cooperation and collaboration in business networks. There are at least three common deployment types of web services and they are all very similar to current business practices. The economics of web services is heavily dependent on fixed costs thus forcing providers to increase the number of users to reach a point of profitability. This may be a decisive factor in using them with ERP solutions where vendors will have to increase their investments to make web services user-friendlier.

1 Introduction

E-business technology gives birth to new ways of cooperation and collaboration for individuals and companies alike. It enables new models of operation, creating numerous new business models and employment opportunities. Entrepreneurs are looking for a business model of the information age that will allow them to identify real options and to assess the consequences of their decisions.

Since the late 1990s, companies have increasingly turned to the Internet and web based technologies to accomplish this goal. The Internet represents in this context an enabling tool for businesses and a new business environment [3]. But what they are finding is that without enterprise resource planning (ERP) software, sharing accurate information with their trading partners is impossible [6]. Further development of ERP solutions even included a lot of different standardized integration technologies, like EDI (Electronic Data Interchange) for example that made information exchange easier.

After different models of e-business emerged, connections between trading partners that were mostly based on 1:1 connections became obsolete. Companies started turning from dedicated connections to the one giving them more flexibility in changing their business partners, the n:n connection.

This change into a new era of cooperation has become a must in a recent year, when a set of new technologies called web services has been introduced. In the year 2002 almost every ERP vendor introduced it's own implementation of web services. Now businesses are again facing a question of whether, how and when they should embrace this technology.

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 177–179, 2003.

2 The Business Value of Web Services

Web Services are a set of technologies including SOAP[1] (Simple Object Access Protocol), UDDI[2] (Universal Description, Discovery and Integration) and WSDL (Web Services Description Language). Listed standards form a technical infrastructure and another set of standards (ebXML[3], RosettaNet[4] etc.) that are mostly in their infancy form a business-oriented framework of web services.

Currently there are at least three common deployment types. The most obvious is to add a web service interface onto an existing product. Of course there is also the complementary use when customers deploy vendor products to solve current integration needs. There is also a third type of almost forgotten application service providers (ASPs), when ASP offers a web service interface so customers can access and use it. There is an interesting difference between these deployment types in the different economic models they imply. Vendors gain revenue generated from sales of their product, customers gain a return on their investment (ROI) from increased efficiency and expanded customer revenue and ASPs make money from recurring or rental revenue of web service itself.

All in all we could say that web services are not especially new. Of course we have seen all this economic models before but were they as successful as we would have hoped for? If they were so successful vendors would have been engaged in other areas of development right now and different standardising bodies would not have been created. The problem lies in the middleware. Almost every business and individual in the developed world has access to the Internet. That in it's essence makes it very easy and cost effective to send some kind of data between collaborating partners whether being businesses or individuals. Cooperation is not limited to a shopping relationship but becomes true collaboration, i.e. the processes work together [5].

Machine to machine interaction is even more complex. A human can quickly adapt to another format of a web page but a computer cannot extract information from different data formats automatically. That is why some vendors decided to make new software tools that would simplify this process. Those tools are known as integration servers and are quite costly. They are mostly used in big companies because smaller ones cannot afford them. Expressed in numbers estimates place the costs of building just one point-to-point connection at $75,000 [2]. The primary business value that we can expect from web services is therefore lowering the cost of integration. In the short term we should see that the businesses will start to get much more value out of their integration and in a long-term very interesting new business models may arise [1].

Before a business can decide to embrace a new technology some positive business indicators have to exist. In the case of web services a business may use or provide some number (Q) of web service based RPC (Remote Procedure Call)

[1] http://www.w3.org/2000/xp/Group
[2] http://www.uddi.org/
[3] http://www.ebxml.org/
[4] http://www.rosettanet.org/

calls in a given period. For a web service provider this will generate a certain amount of total costs (TC) that can be separated in fixed costs (FC) and variable costs (VC). Division of total costs with quantity gives average total costs (ATC) (equation 1).

$$ATC = AFC + AVC \tag{1}$$

With increasing quantity average fixed costs (AFC) quickly decrease. Because material costs and connection costs per RPC are negligible, variable costs can be ignored meaning average total costs equal average fixed costs [4]. In the case of web services average total costs will decrease with increased number of RPC calls meaning every business will strive to increase this number. Therefore web services will become profitable when cumulative number of RPC calls per web service attains a certain number of calls in a given period. We can conclude that they will need to reach a certain critical mass before turning into a profitable product.

3 Conclusions

After testing web services with SAP R/3 as one of the ERP implementations we identified some problems in existing intra- and inter-application integration framework relying on business objects and their methods, BAPIs (Business APIs). Even if SAP AG ensures these can be exposed as web services they are still very complicated to use and too inconsistent. Therefore business partners trying to use web services exposed in such a way would find them very difficult and complex to implement. That would increase ATC costs and require higher output quantity (higher number of RPC calls) to achieve profitability.

ERP vendors will have to show originality in providing support for web services based on different business framework standards. Only then will they be open enough to participate in business networks based on different business busses.

Further study of different ERP implementations should focus on compatibility between different products and platforms.

References

1. Ashton, H. The current state of Web Services. From the mouths of experts. *SearchWebServices*.
2. Fritz, F. J. 2002. Inside edge. *SAP Insider*. (Apr).
3. Gascoyne, R. J. 1997. *Corporate Internet Planning Guide. Aligning Internet Strategy With Business Goals*, Van Nostrand Reinhold.
4. Kezmah, B. 2001. Economics of collaboration. *Information Society 2001. Institut Jožef Stefan* (Ljubljana).
5. Oesterle, H., Fleisch, E. and Alt, R. 2000. *Business Networking. Shaping Enterprise Relationships on the Internet*, Springer-Verlag, Berlin.
6. Norris, G., Hurley, J. R., Hartley, K. M., Dunleavy, J. R. and Balls, J. D. 2000. *E-Business and ERP. Transforming the Enterprise*. John Wiley & Sons, Inc., New York.

An Infrastructure to Manage Resources

Frank Wagner

Roskilde University
Universitetsvej 1
4000 Roskilde, Denmark
frankw@ruc.dk

Abstract. One of the more diffuse problems with information technology (IT) is that both users and their IT-supporters and IT-administrators have the feeling that IT leads to more work to be done. Since the boom of the WWW this got even worse. On one hand we have an easy way of making information available, on the other hand we have all the maintenance work with it. The problem is that much of the information to be published on the web actually already exists somewhere else, just not in the right format. This leads to additional work, inconsistency and other well known problems.
During 1999 I built a prototype to try out some ideas that should help me to manage these problems. The background for my work and this paper is practical. But through the last years my interest in a general understanding of this problem has increased. This position statement is meant both as a status and as a starting point.

1 Introduction

In Sect. 2 I describe the problem and give a little background for my work. In Sect. 3 I try to sketch some perspectives that appeared while working on an improved implementation of the original prototype and especially while writing this paper. In Sect. 4 I describe the concept of the infrastructure and some ideas for an implementation.

2 Background

2.1 What Is the Problem?

The problem seems to be a combination of several irritations. It is difficult to reuse information already in the computer [3]. Some routine work should be done by the computer. There is too much work required to set up the computer. There is much work unrelated to what I actually want to do. These irritations are not bound to specific tasks.

For any user this is irritating, but he often gets used to it. For a supporter it is frustrating to see different users having the same kind of problems. The system administrator can make more tools or systems available, but with such a diffuse problem it is hard to find a point to start (if you don't want to standardize too much).

P.J. Nürnberg (Ed.): MIS 2002, LNCS 2641, pp. 180–185, 2003.

2.2 A Simple Test System

When this really started to irritate me some 4 years ago, we didn't have resources for fundamental changes and these would not have been considered really neccessary by most users. As a consequence I started to work on a small prototype that should test and demonstrate the following ideas: I grab information as early as possible and save it as xml-documents. I choose a few kinds of information and define document types for them. The documents of a certain type are manipulated by xslt-stylesheets specific for this type. I call the combination of document type and related stylesheets for infotype. Information or pieces of it are identified by an id that relates it to its infotype. A few scripts handle the requests to show, update, index and query information. These scripts are independent of the infotypes. They choose the right stylesheets based on the requested id.

The actual case is the lecture catalogue for our department. We collect information about the coming activities (courses, lectures, ...). The information consists basically of a title, a description, related persons, dates and locations and subordinate activities (meetings, seminars, ...). This information is saved and indexed. The index can be shown and information about activities can be accessed by requesting IDs. A special script queries the index for activities and builds a new index by date which is used for a calendar view. If the locations are referenced by an id, they can be accessed from both the information about an activity and the calendar view.

2.3 Did It Work?

Well, part of it (the online lecture catalogue and calendar) has been used since summer 1999. A map over the building works, but never looked good enough to be used. The biggest problem was that the actual implementation of the prototype uses Microsoft's active-server-pages and an early xslt-processor (msxml, pre xslt 1.0), so instead of extending the prototype with more infotypes, I started working on a new implementation. More about it in Sect. 4.

3 Perspectives

About the same time as I got my prototype up and running I started to attend some conferences and workshops (WebNet'99, OHS 5 and HT'99, ...,I-KNOW'02 and now MIS'02). In this section I focus on some for me important terms and concepts. They come from different fields, so to make them work together I use them in a rather naive, common sense way, still using inspiration from the different fields.

3.1 Tool or Assistant?

In Sect. 1 I state that the problem actually is a combination of several irritations or problems not bound to specific tasks. Maybe I can find a perspective that makes it easier to see.

Usually we look at computers as tools. We are the users. We know the tasks and we have the intentions, the computer helps us do it. If we have a well defined combination of tasks, we can describe this combination as a process and build a tool to handle it. This takes responsibility from us as the user and takes care of some problems, but the prize is less flexibility. It only works if we have a well defined process and if it is not too complex.

What if we look at the computer as an assistant? An assistant is supposed to help us actively to do some work. We communicate with an assistant about some work to be done. We establish a common understanding of what we work with. The assistant might use tools, but they are not my problem anymore. I can tell my assistant to collect all dates, no matter whether they belong to a meeting, an incoming e-mail, an event. Then I can ask him what happened at a certain date. Without an assistant, I would have to remember to start my calendar tool everytime, I would have to find logfiles, ...

3.2 Resources and Representations

The term "resource" is borrowed from RFC 2396 which defines Uniform Resource Identifiers (URIs). Sect. 1.1: "A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. ...". Roy Fielding's keynote [2] at the WebNet'99 triggered this choice. He used a simple communication model to illustrate the idea behind URIs and the http-protocol. In my informal words: I know something, I want to share it with you, I represent it in a way (words, pictures, ...) which is relevant for you. I make these representations accessible to you and hopefully you can use them.

I use *Resource* for anything I focus on. I choose a *Representation* of this resource that is appropriate to work with. More about representations in Sects. 3.3 and 3.4.

3.3 Knowledge, Behavior and Representations

The I-KNOW'02 was another key event for me. The final keynote from Robert Trappl [5] triggered some interesting associations. *Knowledge* and *Behavior* are two views of the same thing. Behavior is active, a way to experience and influence. Knowledge is passive, it is the accumulation of experiences. Behavior builds and manipulates knowledge, knowledge is what behavior itself is build on. When the relation is obvious we call it rational behavior. Otherwise we call it emotional behavior. When we focus on something, think about it, work with it, we use and extend our knowledge about it.

3.4 Structure, Data and Representation

So representations are very important for me. What are the characteristics and qualities of a representation? Which structure and which data do I use for a

representation? These questions are very general, so I only pick a few points. Traditionally the focus was on the data. Computers were used for data processing. This has changed. We now talk about information technology and in the field of structural computing structure is considered to be most important [4].

There seem to be different strategies for choosing representations. One is to go for representations that are related to universal models and therefore have a rather complex structure. I prefer representations that are adapted to specific purposes, having simple structures. I expect that I am going to use the representations in a local context. So it will be easier to use a representation optimized for local use. If I need it in a global context I can transform it. A similar approach is discussed in [1].

4 Resource Manager

The basic idea is that I want to see the computer as an assistant or playing the role of an assistant. The strength of this assistant comes from the qualities of the computer. The following description is not even pseudocode. It is meant to illustrate how I want to use what I wrote about in Sect. 2 and 3.

4.1 My Assistant and Me

The first job for my assistant should be to help me manage the resources I work with and the representations of these resources. If I want the computer to be an assistant for me, I have to give it the concept of a resource and the ability to exchange representations. How does a concept for a resource look like for a computer? Well, the concept of a resource is a resource itself, so I need some representations of it. What makes a resource a resource is that it has an identity, it is identified by a URI. Another point is that it is represented by representations. A resource is related to other resources in certain ways. So far I have given the computer only passive knowledge. It needs to know a behavior to become active. We do this by giving it specialized representations the processor can execute. What is the basic behavior? It should be at least the ability to initialize itself. A resource should be able to give access to its representations. Building on this resource defining the basic resource type, new resource types can be defined. Other resource types i need to get started are a resource manager (-resource type) with the ability to manage other resources and a representation (-resource type) with the ability do manipulate representations. This is like an object oriented approach extended to different kinds of representations.

4.2 We and the Rest

Until now I have talked about this basic resource manager and some resources and their representations as I need them, so my and the computer's needs have been decisive with regard to the choice of the representations (structure and data). This is supposed to be my assistant (the computer running the resource

manager) and my own shared base. This might keep me busy for a while, but it won't be enough, so how do I continue? I will need more kinds of representations.

Let's say that I want to use a certain tool. This tool might use its own document format. I consider this document format to be a certain type of representation. If I am lucky, I can extract some information from such a document, if not I at least know where it belongs. In the most extreme case with only opaque documents (or someone who does not want to cooperate with the assistant at all), my resource manager would behave like a kind of file system. It was actually one of intentions that I should be able to use all existing documents by importing documents from a file system.

If I want to present a resource to someone, I might want to use a html-, pdf- or rdf-representation. If the information already exists in other representations, I might be able to generate the new representations from the existing ones by a kind of transformation. Once I have defined this transformation, it should work for all resources that build on the resource type I have defined the representation for.

I have used 'I' most of the time. It could be any entity, for example a group (a Knowledge Node [1]?). It should be easy to work with a shared resource manager as it is designed for communication. Different members of the group could work with different types of representations (text, graphic, code), always having access to related material.

5 Conclusion

I believe there is a chance that a change of perspective can make work with computer more effective. There is much work to be done to make the statements in this paper more reasonable, but I hope to be able to use it as a starting point and comments are welcome. I will try to implement a minimal resource manager as described in Sect. 4. I expect to need only a few Java-classes, xslt-stylesheets and xml-schemas to get started. I hope this can be an example for an infrastructure that gives transparent access to all kinds of tools by considering tools to be implementations of behaviors.

Acknowledgments

My list of references is too short (well, some may be happy not to be mentioned here). I would like to thank at least the OHS working group: the meetings and proceedings have been very inspiring and so was MIS'02.

References

1. Bonifacio, M., Bouquet, P. and Cuel, R. 2002. Knowledge Nodes: the building blocks of a distributed approach to knowledge management. Keynote at *I-KNOW '02 - 2nd International Conference on Knowledge Management* (Graz, Austria, Jul).

2. Fielding, R. T. 1999. Human communication and the design of the modern web architecture. Keynote at *WebNet 99 - World Conference on the WWW and Internet* (Honolulu, HI, Oct).
3. Goyal, S., Reich, S., Behrendt, W. 2003. EMMOs - Enhanced Multimedia Meta Objects for re-purposing of knowledge assets. (this volume).
4. Nürnberg, P. J., Leggett, J. J. and Schneider, E. R. 1997. As we should have thought. *Proceedings of the Eighth ACM Conference on Hypertext (HT '97)* (Southampton, UK, Apr).
5. Trappl, R. 2002. The crucial role of emotions in intelligent software agents. Keynote at *I-KNOW '02 - 2nd International Conference on Knowledge Management* (Graz, Austria, Jul).

Author Index

Arotaritei, Dragos 148
Avramidis, Dimitris 49

Bećarević, Damir 132
Behrendt, Wernher 155
Bendix, Lars 171

Chen, Yan 90
Christodoulakis, Dimitris 49

Dalcher, Darren 58
Davis, Hugh 118

Ekdahl, Bertil 100

Fan, Xinyuan 90

Gordon, Dorrit H. 76
Goyal, Sunil 155
Griffiths, Jon 118

Haake, Jörg M. 18
Hicks, David L. 112

Jiao, Jian 90

Karousos, Nikos 42
Kezmah, Bostjan 177
Kyriakopoulou, Maria 49

Michaelides, Danius T. 118
Millard, David E. 118

Nürnberg, Peter J. 1, 6, 148

Panaretou, Ioannis 42
Pandis, Ippokratis 42

Reich, Siegfried 155
Roantree, Mark 132
Rozman, Ivan 177
Rubart, Jessica 18

Stamou, Sofia 49

Tochtermann, Klaus 29
Tzagarakis, Manolis 42, 49

Wagner, Frank 180
Wang, Dongmin 90
Wang, Weigang 18
Weal, Mark J. 118
Whitehead, E. James, Jr. 76
Wiil, Uffe Kock 9

Zeid, Amir 161