

Lecture Notes in Computer Science

1728

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

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

Jacky Akoka Mokrane Bouzeghoub
Isabelle Comyn-Wattiau
Elisabeth Métais (Eds.)

Conceptual Modeling – ER '99

18th International Conference
on Conceptual Modeling
Paris, France, November 15-18, 1999
Proceedings



Springer

Series Editors

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

Volume Editors

Jacky Akoka
Conservatoire National des Arts et Métiers
292 Rue Saint Martin, F-75141 Paris Cedex 03, France
E-mail: akoka@cnam.fr

Mokrane Bouzeghoub
Elisabeth Métais
Université de Versailles
45 Avenue des Etats Unis, F-78035 Versailles Cedex, France
E-mail: {Mokrane.Bouzeghoub, Elisabeth.Metais}@prism.uvsq.fr

Isabelle Comyn-Wattiau
ESSEC
Avenue Bernard Hirsch, F-92021 Cergy Cedex, France
E-mail: wattiau@essec.fr

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Conceptual modeling : proceedings / ER '99, 18th International Conference on Conceptual Modeling Paris, France, November 15 - 18, 1999. Jacky Akoka ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1728)
ISBN 3-540-66686-9

CR Subject Classification (1998): H.2, H.4, F.4.1, I.2.4, H.1, J.1

ISSN 0302-9743

ISBN 3-540-66686-9 Springer-Verlag Berlin Heidelberg New York

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

© Springer-Verlag Berlin Heidelberg 1999
Printed in Germany

Typesetting: Camera-ready by author
SPIN 10703309 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

Preface

This book provides a comprehensive state-of-the-art, in conceptual modeling. It grew out of research papers presented at the 18th International Conference on Conceptual Modeling (ER '99) and arranged by the editors. The plan of the conference is to cover the whole spectrum of conceptual modeling as it relates to database and information systems design and to offer a complete coverage of data and process modeling, database technology, and database applications. The aim of the conference and of these proceedings is to present new insights related to each of these topics. This book contains both selected and invited papers. The 33 selected papers are organized in 11 sessions encompassing the major themes of the conference, especially :

- schema transformation, evolution, and integration
- temporal database design
- views and reuse in conceptual modeling
- advanced conceptual modeling
- business process modeling and workflows
- data warehouse design.

Besides the selected papers, 3 invited papers present the views of three keynote speakers, internationally known for their contribution to conceptual modeling and database research and for their active role in knowledge dissemination. Peter Chen presents the results of his ongoing research on ER model, XML, and the Web. Georges Gardarin presents the first results of an ESPRIT project federating various data sources with XML and XML-QL. Finally, Matthias Jarke develops a way to capture and evaluate the experiences gained about process designs in so-called process data warehouses.

The program of ER '99 was designed as a wide forum for researchers and industrial experts interested in all aspects of conceptual modeling. As a result, the program a scientific conference, three workshops, three invited talks, four tutorials, an industrial track, posters, and exhibitions. The research track included 33 papers selected from 143 papers submitted for review. These papers originated from 36 countries from all continents. The three international workshops incorporated 31 papers selected from 65 submitted. They explore the following topics: reverse engineering in information systems, World-Wide Web and conceptual modeling, as well as evolution and change in data management. Besides the invited talks, four tutorials given by internationally recognized experts provided surveys respectively on Web-based information access, visual interactions with databases, UML from analysis to implementation, and designing large scale client-server systems with UML. The industrial track is devoted to presentations by industrial experts. Finally, posters and exhibitions allowed researchers to present ongoing research.

The organization of the conference involved many partners and valuable contributions from a number of persons. First we reserve our greatest debt of gratitude for all those who contributed to the success of ER '99, the authors, invited speakers, tutorial presenters, session chairs and participants. This conference was made possible thanks to the financial and scientific support of the CEDRIC laboratory of the Conservatoire

National des Arts et Métiers, ESSEC Graduate Business School, PRiSM laboratory of the University of Versailles, Institut National de la Recherche en Informatique et Automatique (INRIA), Institut National des Télécommunications (INT), and Università di Roma la Sapienza. They deserve our deepest thanks. We appreciate the help provided by Maurizio Lenzerini who accepted to chair the conference. We would like to thank in particular Stephen Liddle for providing an good software platform for paper and review management. Our thanks goes to program and organization committee members, especially Zoubida Kedad and Nadira Lammari for their technical support. Special thanks goes to Christine Parent and Jacques Kouloumdjian for their untiring efforts in organizing respectively the tutorials and the workshops. We would like to thank David Avison for handling the publicity of ER '99. Finally, we would like to express our sincere appreciation to Isabelle Comyn-Wattiau and Elisabeth Métais. Their enthusiasm and their dedication have made this conference possible.

September 1999

J. Akoka, M. Bouzeghoub

Conference Organization

Conference Chair

Maurizio Lenzerini, University of Rome - La Sapienza, Italy

Program Co-chairs

Jacky Akoka, CNAM & INT, Paris, France

Mokrane Bouzeghoub, University of Versailles, France

Organization Co-chairs

Isabelle Comyn-Wattiau, University of Cergy & ESSEC, France

Elisabeth Métais, University of Versailles, France

Organization Committee

Franck Billot, ESSEC, France

Patricia Borgo, ESSEC, France

Dominique Briolat, ESSEC, France

Chantal Ducoin, PRISM, France

Zoubida Kedad, PRISM, France

Nadira Lammari, CNAM, France

Michel Maurice-Demourieux, INT, France

Tutorial Chair

Christine Parent, Switzerland

Panel Chair

Bernhard Thalheim, Germany

Workshops Chair

Jacques Kouloumdjian, France

Publicity Co-chairs

David Avison, UK

Ramez El Masri, USA

Area Liaison

Asia: Tok Wang Ling, Singapore

Australia: John Roddick, Australia

South America: Alberto Laender, Brazil

North America: Sudha Ram, USA

Europe: Stefano Spaccapietra, Switzerland

Steering Committee Liaison

Kathi Hogshead Davis, USA

Steering Committee Representatives

Chair: Bernhard Thalheim, Germany

Vice-Chair: Tok Wang Ling, Singapore

Emeritus: Peter P. Chen, USA

Tutorials

Web-Based Information Access

by Tiziana Catarci, Università degli Studi di Roma "La Sapienza", Italy

Visual Interaction with Databases

by Tiziana Catarci, Università degli Studi di Roma "La Sapienza", Italy

UML at Work - From Analysis to Implementation

by Gerti Kappel, Martin Hitz, Werner Retschitzegger, Austria

Designing Large Scale Client/Server Systems with UML

by Leszek A. Maciaszek, Macquarie University, Sydney, Australia

Workshops

Workshop 1 - REIS

International Workshop on Reverse Engineering in Information Systems

Chair - J. Kouloumdjian

Workshop 2 - WWWCM

International Workshop on the World-Wide Web and Conceptual Modeling

Co-chairs - P.P.S. Chen, D. Embley, S. Liddle

Workshop 3 - ECDM

International Workshop on Evolution and Change in Data Management

Chair - J. Roddick

Program Committee

Michel Adiba (France)	Daniele Nardi (Italy)
Valeria de Antonellis (Italy)	Erich Neuhold (Germany)
Paolo Atzeni (Italy)	Yves Pigneur (Switzerland)
David Avison (UK)	Alain Pirotte (Belgium)
Nacer Boujlida (France)	Sudha Ram (USA)
Tiziana Catarci (Italy)	John Roddick (Australia)
Corine Cauvet (France)	Colette Rolland (France)
Roger H.L. Chiang (Singapore)	Klaus-Dieter Schewe (Germany)
Isabelle Comyn-Wattiau (France)	Michel Scholl (France)
Jean-Claude Courbon (France)	Arne Solvberg (Norway)
Kathi Hogshead Davis (USA)	Stefano Spaccapietra (Switzerland)
Klaus Dittrich (Switzerland)	Veda Storey (USA)
Phillip Ein-Dor (Israël)	Bernhard Thalheim (Germany)
Ramez Elmasri (USA)	Dimitri Theodoratos (Greece)
David Embley (USA)	Patrick Valduriez (France)
André Flory (France)	
Enrico Franconi (UK)	
Antonio Furtado (Brazil)	
Georges Gardarin (France)	
Henri Habrias (France)	
Jean-Luc Hainaut (Belgium)	
Terry Halpin (USA)	
Igor Hawryszkiewicz (Australia)	
Uwe Hohenstein (Germany)	
Matthias Jarke (Germany)	
Paul Johannesson (Sweden)	
Yahiko Kambayashi (Japan)	
Hannu Kangassalo (Finland)	
Jacques Kouloumdjian (France)	
Alberto Laender (Brazil)	
Marie-Christine Lafaye (France)	
Régine Laleau (France)	
Mong Li Lee (Singapore)	
Stephen Liddle (USA)	
Tok Wang Ling (Singapore)	
Frederick Lochovsky (China)	
Pericles Loucopoulos (UK)	
Leszek Maciaszek (Australia)	
Stuart Madnick (USA)	
Heinrich Mayr (Austria)	
Elisabeth Métais (France)	
Michele Missikoff (Italy)	

Additional Referees

Youssef Amghar
 Pascal André
 Luciano Baresi
 Andreas Behm
 Camille Ben Achour
 Khalid Benali
 Ralph Busse
 Diego Calvanese
 Dunren Che
 Yanjung Chen
 Angelo E. M. Ciarlini
 Ernesto Damiani
 Irini Dimitromanolaki
 Thomas Feyer
 Daniela Florescu
 Chiara Francalanci
 Matthias Friedrich
 Hans Fritschi
 Agnès Front
 Maria Grazia Fugini
 Irini Fundulaki
 Giuseppe De Giacomo
 Georgios Giannopoulos
 Jean-Pierre Giraudin
 Christophe Gnaho
 Mohand-Saïd Hacid
 Wynne Hsu
 Panos Kardasis
 Vagelio Kavakli
 Thomas Klement
 Markus Kradolfer
 Frederique Laforest
 Nadira Lammari
 Michel Leclere

Joseph Lee
 Maurizio Lenzerini
 Jana Lewerenz
 Lucas Mortimer Macedo
 Patrick Marcel
 Paolo Merialdo
 Laurent Mignet
 Gilles Nachouki
 Laurent Nemirovski
 Selmin Nurcan
 Norman Paton
 Ilias Petrounias
 Evaggelia Pitoura
 Roberto Pizzicannella
 George Politis
 Danny Poo
 Nikos Prekas
 Rodolfo F. Resende
 Thomas Risse
 Claudia Roncancio
 Riccardo Rosati
 Giuseppe Santucci
 Martin Schönhoff
 Altigran Soares da Silva
 Eng Koon Sze
 Hee Beng Kuan Tan
 Sergio Tessaris
 Dimitrios Tombros
 Genoveva Vargas Solar
 Athanasios Vavouras
 Scott N. Woodfield
 Irene Woon
 Nur Yilmazturk
 Xiangru Yuan

Table of Contents

Session 1 - Supporting Schema Evolution

Three Levels of Reuse for Supporting the Evolution of Database Schemas	1
<i>D. Castelli, Italy</i>	

A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases	16
<i>B. Benatallah, Australia</i>	

An Incremental and Semi-automatic Method Inheritance Graph Hierarchy Construction	31
<i>M.M. Gammoudi, I. Nafkha, Z. Abdelouahab, Tunisia, Brazil</i>	

Session 2 - Temporal Database Design

Developing an Object-Oriented Video Database System with Spatio-Temporal Reasoning Capabilities	47
<i>S.S.-M. Chan, Q. Li, China</i>	

Entity Evolution in IsA Hierarchies	62
<i>A. Olivé, D. Costal, M.-R. Sancho, Spain</i>	

Temporal ER Modeling with Description Logics	81
<i>A. Artale, E. Franconi, United Kingdom</i>	

Session 3 - Schema Transformation

Automatic Migration and Wrapping of Database Applications – A Schema Transformation Approach	96
<i>P. McBrien, A. Poulovassilis, United Kingdom</i>	

A Methodology for Clustering Entity Relationship Models – A Human Information Processing Approach	114
<i>D.L. Moody, A. Flitman, Australia</i>	

Designing Good Semi-structured Databases	131
<i>S.Y. Lee, M.L. Lee, T.W. Ling, L.A. Kalinichenko, Singapore, Russia</i>	

Session 4 - Views and Conceptual Modeling

Building Views over Semistructured Data Sources	146
<i>S. Castano, V. De Antonellis, Italy</i>	

Modeling and Maintaining Multi-view Data Warehouses	161
<i>I. Stanoi, D. Agrawal, A. El Abbadi, USA</i>	

Object Views through Search Views of Web Datasources	176
<i>Z. Lacroix, USA</i>	

Session 5 - Reuse in Conceptual Modeling

Understanding and Modelling Business Processes with DEMO	188
<i>J. L. G. Dietz, The Netherlands</i>	

A Methodology for Building a Repository of Object-Oriented Design Fragments ..	203
<i>T.-D. Han, S. Purao, V.C. Storey, USA</i>	

An Algorithm to Extract Is_A Inheritance Hierarchies from a Relational Database	218
<i>N. Lammari, France</i>	

Session 6 - Business Process Modeling and Workflows

DOTS: A Generic Infrastructure for Decision-Oriented Collaborative Task Support	233
<i>J. Lonchamp, France</i>	

Dockets: A Model for Adding Value to Content	248
<i>J. W. Schmidt, H.-W. Sehring, Germany</i>	

Workflow Specification in TRAMs	263
<i>M. Kradolfer, A. Geppert, K.R. Dittrich, Switzerland</i>	

Session 7 - Integrating Application Models

From CASE to CARE (Computer-Aided Requirements Engineering)	278
<i>O. Pastor, J.H. Canós, I. Ramos, Spain</i>	

Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators	293
<i>V. M. P. Vidal, B.F. Loscio, Brazil</i>	

A Method for Requirements Elicitation and Formal Specification	309
<i>M. Heisel, J. Souquières, Germany, France</i>	

Session 8 – Data Warehouse Design

Dealing with Semantic Heterogeneity During Data Integration	325
<i>Z. Kedad, E. Métais, France</i>	

Detecting Redundancy in Data Warehouse Evolution	340
<i>D. Theodoratos, Greece</i>	

Modelling Data Warehouses and OLAP Applications by Means of Dialogue Objects	354
<i>J. Lewerenz, K.-D. Schewe, B. Thalheim, Germany</i>	

Session 9 - Modeling Concepts

Resolving the "Weak Status" of Weak Entity Types in Entity-Relationships Schemas	369
<i>M. Balaban, P. Shoval, Israel</i>	

A Taxonomy of Recursive Relationships and Their Structural Validity in ER Modeling	384
<i>J. Dullea, I.-Y. Song, USA</i>	

Extending Functional Dependencies in Indefinite Sequence Relations	399
<i>W. Ng, China</i>	

Session 10 - Schema Integration

View Integration of Object Life-Cycles in Object-Oriented Design	413
<i>G. Preuner, S. Conrad, Austria, Germany</i>	

Towards an Automatic Integration of Statecharts	430
<i>H. Frank, J. Eder, Austria</i>	

Design Support for Database Federations	445
<i>K. Schwartz, I. Schmitt, C. Türker, M. Höding, E. Hildebrandt, S. Balko, S. Conrad, G. Saake, Germany</i>	

Session 11 - Advanced Conceptual Modeling

Operational Characterization of Genre in Literary and Real-Life Domains	460
<i>A.L. Furtado and A. E. M. Ciarlini, Brazil</i>	

Contextualization as an Abstraction Mechanism for Conceptual Modelling	475
<i>M. Theodorakis, A. Analyti, P. Constantopoulos, N. Spyros, Greece, France</i>	

Architectures for Evaluating the Quality of Information Models – A Meta and an Object Level Comparison	490
<i>R. Schuette, Germany</i>	

Invited Papers

XML-based Components for Federating Multiple Heterogeneous Data Sources 506
G. Gardarin, F. Sha, T.D. Ngoc, France

A Process-Integrated Conceptual Design Environment for Chemical Engineering ..520
M. Jarke, T. List, K. Weidenhaupt, Germany

ER Model, XML and the Web 538
P.P.S. Chen, USA

Author Index 539

Three Levels of Reuse for Supporting the Evolution of Database Schemas

Donatella Castelli

Istituto di Elaborazione della Informazione-CNR, Pisa, Italy
castelli@iei.pi.cnr.it

Abstract. This paper presents a reuse technique which permits to simplify redesign that may occur when the conceptual schema is changed. This technique is based on the reuse of the refinement steps, the verifications and the schemas of the original design. For each kind of reuse, the technique specifies the conditions under it can be applied and how the reused components should evolve.

1 Introduction

When a conceptual schema evolves the whole design process has to be re-examined to adapt it to the new situation. In the database design practice it is often recommended to manage this evolution by employing methods based on the reuse of previous design [1,3,11,14,4]. The use of these methods, however, is far from being trivial. Often, a significant effort is required to select the appropriate components to be reused and to understand which modifications must be operated. This effort, on the one hand discourages the use of these methods, and on the other reduces the advantages of reuse.

This paper shows how this difficulty can be overcome when the schema design is interpreted through schema transformations. To this aim, it proposes precisely defined database schema design maintenance techniques. These techniques specify the conditions under which reuse can be applied and how the different reused components evolve in response to the changes applied to the conceptual schema.

The techniques illustrated are embedded within the transformational framework described in [10]. Several other frameworks [1,3,2], either implicitly or explicitly transformational, can be rephrased in terms of this one. As a consequence, the results presented in this paper can be easily extended also to all these other frameworks.

The techniques proposed is introduced by progressively increasing the amount of reuse. Section 3, introduces the reuse technique. Section 4 focuses on the reuse the refinement steps. Refinement steps can be reused provided that they still produce a correct design process. Section 4 shows how the reuse of proofs can be exploited to reduce the cost of this check for correctness. The techniques presented are then generalised in Section 5 to include also the reuse of the schemas that document the design. In particular, the modifications to be applied to each

of the schemas of the previous design are precisely characterised. Finally, concluding remarks are given in Section 6. The next Section introduces the schema design transformational framework that will be used in the rest of the paper.

2 A Framework for Database Design

The transformational framework that will be used to illustrate the proposed reuse technique is very simple and general [9]. It has been preferred to others since its characteristics permit to derive results that can be easily transferred to other database design frameworks.

In this framework the whole design relies on a single notation [8]. This notation, illustrated briefly through the example in Figure 1, allows to model the database structure and behavior into a single module, called *Database Schema* (DBS). This module encloses classes, attributes, is-a relations, integrity constraints and operation specifications.

```

database schema StoneDB
class marbles of MATERIAL
  with (close_up_photo:IMAGE; long_distance_photo:IMAGE)
class for_exterior_m is-a marble
  with (e_colour: COLOUR)
class for_interior_m is-a marble
  with (i_colour: COLOUR)
constraints
  e_colour = { (x,y) · x ∈ for_exterior_m ∧ ∃k (k ∈ long_distance_photo(x) ∧ y = get_colour(k)) }
  i_colour = { (x,y) · x ∈ for_interior_m ∧ ∃k (k ∈ close_up_photo(x) ∧ y = get_colour(k)) }
  for_exterior_m ∪ for_interior_m = marble
initialisation
  marbles, close_up_photo, long_distance_photo,
  for_interior_m, for_exterior_m, i_colour, e_colour := ∅
operations
  y ← return_colour(x) =
    pre x ∈ marble then if x ∈ (for_interior_m - for_exterior_m)
      then y := { i_colour(x) }
    else if x ∈ (for_exterior_m - for_interior_m)
      then y := { e_colour(x) } else y := { i_colour(x), e_colour(x) }
    end
end

```

Fig. 1. A Database Schema

The notation of Database Schemas is formalized in terms of a formal model introduced within the B-Method [5]. This formalization allows to exploit the B theory and tools for proving expected properties of the DBS schemas.

Both the construction of the conceptual schema and the refinement of this schema into the chosen DBMS schema are carried out by means of schema trans-

formations. Table 1 and Table 2 show, respectively, the primitives for schema construction, collectively called *Schema Transformation Language*(STL), and those for schema refinement, called *Schema Refinement Language*(SRL). Each transformation has a further parameter, which has been omitted in the tables for simplicity, which specifies the name of the DBS schema that is transformed. The semantics of each operator can intuitively be understood by the name of the operator. Both the languages comprise also a composition operator “o” which allows to construct more complex transformations [10]. This operator is enough powerful to render the two languages complete. Moreover, an additional composition operator “;” is given to compose transformations sequentially.

add.class (<i>class.id, class.type, class.init</i>)
rem.class (<i>class.id</i>)
mod.class(<i>class.id, class.type, class.init</i>)
add.attr(<i>attr.id, class.id, attr.type, attr.init</i>)
rem.attr (<i>attr.id, class.id</i>)
mod.attr (<i>attr.id, class.id, attr.type, attr.init</i>)
add.isa(<i>class.id1, class.id2</i>)
rem.isa (<i>class.id1, class.id2</i>)
add.constr(<i>constraint</i>)
rem.constr (<i>constraint</i>)
add.op (<i>op.id, body</i>)
rem.op (<i>op.id</i>)

Table 1. STL language

r.add.class (<i>class.id, expr</i>)
r.rem.class (<i>class.id, expr</i>)
r.add.attr(<i>attr.id, class.id, expr</i>)
r.rem.attr (<i>attr.id, class.id, expr</i>)
r.add.isa(<i>class.id1, class.id2</i>)
r.rem.isa (<i>class.id1, class.id2</i>)
r.mod.op (<i>op.id, body</i>)

Table 2. SRL language

The expressions that appear as a parameter in the r.add/rem transformations specify how the new/removed element can be derived from the already existing/remaining ones. These conditions are required since only redundant components can be added and removed in a refinement step. The SRL language does not permit to add or remove constraints and operations. It only permits to change the way in which an operation is defined. This is done both explicitly, through the r.mod.op operator, and automatically, as a side effect of the transformations that add and remove schema components. These automatic modifications add appropriate updates for each of the new schema variables, cancel the occurrences of the removed variables, and apply the proper variable substitutions.

A transformation can be applied when its *applicability conditions* are verified. These conditions, that must be verified before the application of the transformation, prevent from the execution of both meaningless transformations and transformations that can break the schema consistency and the correctness of the design. The applicability conditions are given by the conjunction of simple conditions, called *applicability predicates*. In SRL two kinds of applicability predicates can be distinguished¹:

¹ The specification of the STL applicability predicates is omitted since it is not useful for the rest of the paper.

1. Applicability predicates that prevent from the application of meaningless refinements. These are the predicates requiring that an element be/be not in the input schema (these will be indicated in the rest of the paper with, respectively, $x \in S$ and $x \notin S$). For example, an attribute to be removed has to be in the schema, a class to be added has not to be in the schema.
2. Applicability predicates that prevent from breaking the correctness of the design. These, can be partitioned into:
 - predicates requiring that the expression in a transformation call be given in terms of state variables ($\{x_1, \dots, x_n\} \subseteq S$), where $x_1, \dots, x_n$ are the free variable of the expression). For example, the expression that defines how a new class can be derived from the rest of the schema components must contain only state variables of the input schema;
 - predicates requiring that only redundant elements be removed ($\text{Constr} \Rightarrow x=E$). For example, a class cannot be removed if the removed element cannot be proved to be equivalent to E ;
 - predicates requiring that the DBS inherent constraints be preserved ($\text{Constr} \Rightarrow \text{Inh}$). This means that each time the input is a DBS schema also the result must be a DBS schema. For example, it is not possible to remove a class if there is an attribute that refers this class;
 - predicates requiring that the body of an operation be substituted with an algorithmic refinement of the previous one ($\text{Body}' \sqsubseteq \text{Body}$) (see [5] for the definition of $\sqsubseteq$).

STL and SRL are not independent since each SRL primitive transformation can be expressed as a particular composition of STL transformations. For example, $r.add.class(c, E)$, which adds a redundant class c , is defined as the composition of the STL primitives: $add.class(c, type(E), E)$, $add.constraint(c=E)$, and a set of $mod.op(op, subs)$, one for each operation that must be changed to introduce the appropriate updates for the new class. Similarly, the $r.rem.class(c, E)$ is the composition of $rem.class(c)$, as many $rem.constraint$ and $add.constraint$ as are the constraints in which c must be replaced by E , and as many $mod.op$ as are the operations in which a similar change of variable is needed.

The set of transformations that can be specified using SRL is a subset of those that can be specified using STL. As a matter of fact, STL specifies all and only the schema transformations that, when applied to a consistent schema, produce a consistent schema. SRL specifies exactly the subset of these schema transformations that produce refinements of the input schema, i.e., which produce schemas that satisfy the specification given by the input one. Certainly, these characteristics of the two languages hold under the assumption that the applicability conditions of the transformations are verified.

3 The Propagation of Changes

Figure 2a) shows the schematisation of a database schema design process. In the figure S_0 indicates the conceptual schema. S_0 is transformed into the final

logical schema S_2 through two refinement transformations, t_{r1} and t_{r2} , each of which produces a new schema. Each schema in the sequence contains more implementative details than the previous one. Figure 2b) shows what happen when the conceptual schema evolves as an effect of new requirements. We can schematise this evolution, again, as the application of a schema transformation. Let call this transformation t_0 . The change applied to the conceptual schema is then propagated to the whole design process, i.e. new refinement steps are applied to the schema S'_0 to produce the new final schema and new intermediate schemas, S'_1 and S'_2 , are produced.

In the literature, the above approach to the propagation of changes is considered the more appropriate since, by producing a complete documentation of the new situation, it renders the subsequent modifications easier [6,7]. However, since its cost can be high, it is often recommended to carry it out by employing techniques based on the reuse of the previous design documentation. Referring to our schematisation of the design process, we might think to reuse the refinement steps, the correctness proofs and the intermediate schemas that document the design process. The reuse of the refinement steps may be useful in those situations in which no rethinking on the optimisation decisions and on the DBMS choice is planned. The reuse of the proofs done may be helpful in assessing the correctness of the new design. Finally, the reuse of the schemas may be useful to avoid the regeneration of the schemas that document the design.

Reuse, however, is really effective in reducing the complexity of redesign only if it is clear how the reusable components can be identified and which modifications must be operated to satisfy the new needs. Most of the proposals for database (re)design that rely on reuse, miss this specification [1,3,11,2,15]. This lack renders the application of reuse complex. It also makes difficult to built really useful automatic or semi-automatic redesign supporting tools. The aim of this work is to overcome this drawback by characterising precisely each of the above identified kinds of reuse.

In order to illustrate these different form of reuse a simple case of schema refinement will be used along the whole paper. This consists of a single refinement

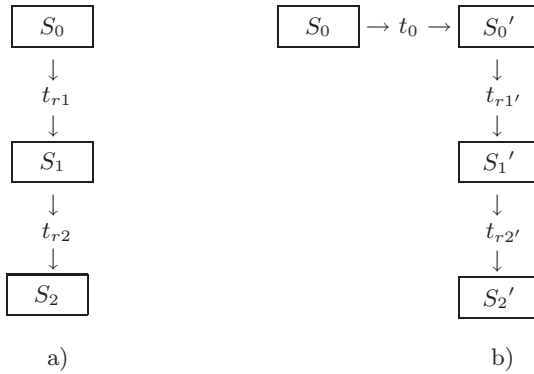


Fig. 2. The reuse of refinement steps

step (see Figure 3). It takes as input the schema given in Figure 1 and produces a schema in which the possibly overlapping *for_exterior_m* and *for_interior_m* classes are replaced with three mutually exclusive classes: *ei*, *e-i* and *i-e*. Moreover, the redundant attributes *i_colour* and *e_colour* are removed.

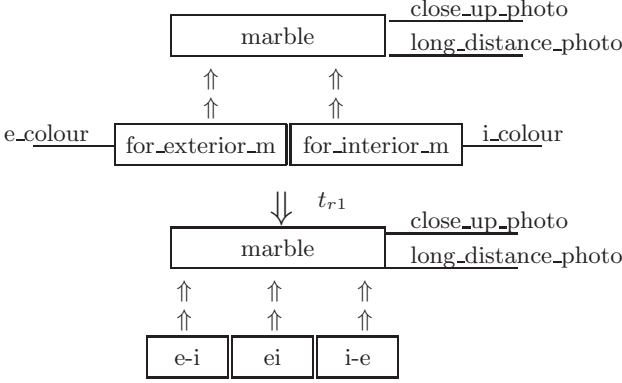


Fig. 3. An example of schema refinement

The transformation t_{r1} that specifies the refinement step² is defined as follows:

```

r.add.class(ei, ei=(for_interior_m ∩ for_exterior_m)) ∘
r.add.class(i-e, i-e=(for_interior_m - for_exterior_m)) ∘
r.add.class(e-i, e-i=(for_exterior_m - for_interior_m)) ∘
r.add.isa(marble, ei) ∘ r.add.isa(marble, i-e) ∘ r.add.isa(marble, e-i) ∘
r.rem.isa(marble, for_interior_m) ∘ r.rem.isa(marble, for_exterior_m) ∘
r.rem.class(for_interior_m, for_interior_m= (ei ∪ i-e)) ∘
r.rem.class(for_exterior_m, for_exterior_m= (ei ∪ e-i)) ∘
r.rem.attr(i_colour, i_colour = {(x,y)·x∈ for_interior_m ∧
                                   ∃ k (k∈close_up_photo(k) ∧ y=get_colour(k))})
r.rem.attr(e_colour, e_colour = {(x,y)·x∈ for_exterior_m ∧
                                   ∃ k (k∈long_distance_photo(k) ∧ y=get_colour(k))})

```

As a side effect of the above refinement transformation the first two integrity constraints become logically equal to true. The third constraint becomes:

$$(e-i \cup ei \cup i-e) = marble$$

and, finally, the *return_colour(x)* operation becomes:

² Here, the refinement transformation is given in terms of elementary transformations. Usually, more complex transformations are used. These are defined as composition of elementary ones and permit a more concise expression of the refinement steps.

```

 $y \leftarrow \text{return\_colour}(x) =$ 
pre  $x \in \text{marble}$ 
then if  $x \in (i-e)$  then  $y := \{\text{get\_colour}(\text{close\_up\_photo}(x))\}$ 
      else if  $x \in (e-i)$  then  $y := \{\text{get\_colour}(\text{long\_distance\_photo}(x))\}$ 
      else  $y := \{\text{get\_colour}(\text{close\_up\_photo}(x)),$ 
           $\text{get\_colour}(\text{long\_distance\_photo}(x))\}$ 
end

```

The applicability conditions for t_{r1} are generated according the rule given in [10]³. Stated informally, these conditions are:

- there are no classes in the previous schema with identifiers $e-i$, ei and $i-e$;
- the is-a links added are not in the previous schema and they do not generate any is-a loop;
- the removed elements belong to the schema;
- the removed classes have no attributes defined on them;
- the removed elements are redundant; i.e. the information they represent can be derived from the remaining component of the schema.

The above conditions can be easily proved to be verified on the initial schema S_0 . This ensures that the output produced by t_{r1} is a refinement of S_0 .

We can now begin our characterisation of the different levels of reuse by considering the most simple case: the reuse of the refinement steps.

4 The Reuse of the Refinement Steps

The reuse of refinement steps is the simpler form of reuse. It is useful when the implementation decisions that have been taken during the original design are valid also for the new design. In this case, the transformation may be either re-applied as it is or he can be extended to specify additional implementational details. The extension may be required, for example, for refining the components that have been added to the conceptual schema. When the reuse of the refinement step is chosen then $t_{ri'}$ is defined either as:

$$t_{ri'} = t_{ri} \quad \text{or} \\ t_{ri'} = t_{ri} \circ t_{rext} \quad \text{where } t_{rext} \text{ is an enrichment transformation.}$$

Referring to our case design, suppose that the initial schema is changed so to include an additional attribute *tech_char* assigned to the class *marble*. The designer may decide that no refinement of the new attribute is needed. In this case, he can reuse the refinement steps t_{r1} as it. Alternatively, he may decide that he want to move the new attribute from the class *marble* to each of the three subclasses. In this case he can simply build the new refinement step as the composition of t_{r1} and the following transformation:

³ Actually, the rule cited generates only the last condition since it is able to automatically discharge the others. Here we have chosen to report the complete set of applicability conditions for a better understandability.

$$t_{next} = r.add.attr(e-i_{tech.char}, e-i) \circ r.add.attr(ei_{tech.char}, ei) \circ \\ r.add.attr(i-e_{tech.char}, i-e) \circ r.rem.attr(tech.char)$$

where the added attributes are those that replace *tech.char*.

The decision to reuse a refinement step has always to be ratified by proving that the chosen refinement step does not contradict the semantics of the schema it is applied to. In our framework this is expressed by requiring that the applicability conditions of the refinement transformation, evaluated on the new schema, be verified. The next Section discusses how this verification can be done by reusing the proofs done during the previous design.

5 The Reuse of the Correctness Proofs

The aim of this Section is to specify which are the applicability conditions that can be discharged by reusing the proofs done in the previous design.

In this discussion we will restrict ourself to consider only the relations among t_0 , S_0 and t_{r1} as defined in Section 3. In the next Section we will see how, by adopting a different view, the whole design can be reduced to a sequence of cases that have the shape that has been just considered. This will permit us to generalise the reusing rules discussed here to the whole design process.

In order to simplify the characterization, it will be assumed that there is no overlapping between t_0 and t_{r1} . This not causes any loss of generality since, in case this overlapping exists, we can think to use, during the replaying of the design, refinement transformations in which the overlapping part has been removed. It will also be assumed that the consistency of $t_0(S_0)$ has been proved and that the previous design is correct.

Let S_0 be defined as $\langle Cl, Attr, IsA, Constr, Op \rangle$ where: Cl is a set of classes, $Attr$ is a set of attributes, IsA is a set of is-a relationships, $Constr$ is a set of integrity constraints, and Op is a set of schema operations. The set Op always contains an operation *Init* that specifies the schema initialisation. According to the semantics given in [9] the application of t_0 to S_0 has the following effect:

$$t_0(\langle Cl, Attr, IsA, Constr, Op \rangle) = \langle Cl \cup AddedCl - RemovedCl, \\ Attr \cup AddedAttr - RemovedAttr, \\ IsA \cup AddedIsA - RemovedIsA, \\ Constr \cup AddedConstr - RemovedConstr, \\ Op \cup AddedOp - RemovedOp \rangle$$

where: *AddedX* and *RemovedX* stands for the set of components of kind X that are, respectively, added and removed from the schema by t_0 . In particular, *Added-Constr* comprises the inherent constraints that are associated to the new elements of the schema. In order to reuse a refinement transformation for carrying out a redesign, we must be ensured that the transformation can still be applied to the modified schema. This means that the applicability conditions of the reused transformation alone, or, in case the reused transformation is extended, those of the composed transformation, have to be proved. Let us consider, first, the

case in which there is no expansion, i.e. the case in which we have to prove the applicability conditions $appl_{t_{r+1}}(S'_0)$ where $S'_0 = t_0(S_0)$. Let us reason on the type of predicates that compose these applicability conditions. Each type of predicate will be indicated with the same label used in Section 2. Remember that we have assumed that the refinement transformations never add and remove the same schema component.

– $x \in Cl$

The following subcases can be distinguished:

- $x \in AddedCl \wedge x \notin RemovedCl$; the predicate is true.
- $x \notin AddedCl \wedge x \in RemovedCl$; the predicate is false.
- $(x \notin AddedCl \wedge x \notin RemovedCl)$; the predicate has to be evaluated on the schema S_0 .

– $x \in Attr, x \in IsA$; these cases are similar to the previous one.

– $x \notin Cl$; The following subcases can be distinguished:

- $x \in AddedCl \wedge x \notin RemovedCl$; the predicate is false.
- $x \notin AddedCl \wedge x \in RemovedCl$; the predicate is true.
- $(x \notin AddedCl \wedge x \notin RemovedCl)$; the predicate has to be evaluated on the schema S_0 .

– $x \notin Attr, x \notin IsA$

; these cases are similar to the previous one.

– $Free(E) \subseteq SchemaVar$

; if t_0 does not remove/add elements then the predicate has to be evaluated on the schema S_0 , otherwise it must be proved on the new schema.

– $Constr \Rightarrow x=E, Constr \Rightarrow Inh$

; if t_0 does not remove/add constraints then the predicate has to be evaluated on the schema S_0 , otherwise it must be proved on the new schema.

– $Op \text{ ref } Op'$

; if Op belongs to $RemovedOp$ then this predicate has not to be proved. In all the other cases, it has to be evaluated on the schema S_0 .

This analysis shows that there are conditions that can be evaluated directly through simple checks, conditions that has to be evaluated on the schema S_0 and conditions that have to be proved on the new schema. The second ones are those that have been proved during the previous design. We can thus reuse the proofs already done to discharge them. The actual “re-applicability conditions”, i.e. those conditions which need to be proved, are only the third ones.

The following two examples show how the reuse of proofs can be fruitfully exploited. These assume as previous design that given at the end of Section 2.

Example 1

The conceptual schema given in Figure 1 specifies that a marble can have associated zero, one or two photographs. Let us imagine that the requirements become stronger so that there has to be at least one close up photograph for each marble suitable for interior use and at least one long distance photograph for each marble suitable for exterior use. In order to be consistent with the new requirements, a new constraint is added to the conceptual schema. This is done through the following schema transformation:

$$t_0 = \text{add.constraint}((\forall m. m \in \text{for\_exterior\_m} \rightarrow \exists k \text{ long\_distance\_photo}(m) = k) \wedge \\ (\forall m. m \in \text{for\_interior\_m} \rightarrow \exists k \text{ close\_up\_photo}(m) = k))$$

Let us imagine that in this redesign we decide to reuse t_{r1} . In order to safely do that, we have to verify that its applicability conditions are verified on the new schema. The new applicability conditions differs from those already proved during the previous design only because of the last condition that has a richer antecedent. This difference does change the truth value of this condition. The new applicability conditions can thus be discharged completely by relying on the proofs done previously.

Example 2

As another example, let us suppose that the colour of marbles is not meaningful anymore. All the reference to this information has to be removed from the conceptual schema. The following is the definition of the transformation t_0 that does the job. It cancels the attributes i_colour and e_colour and the schema components that refer them.

$$\begin{aligned} & \text{rem.attr}(i\_colour, \text{for\_interior\_m}) \circ \text{rem.attr}(e\_colour, \text{for\_exterior\_m}) \circ \\ & \text{rem.constr}(e\_colour = \{(x, y). x \in \text{for\_exterior\_m} \wedge \\ & \quad \exists k \cdot (k \in \text{long\_distance\_photo}(k) \wedge y = \text{get\_colour}(k))\}) \circ \\ & \text{rem.constr}(i\_colour = \{(x, y). x \in \text{for\_interior\_m} \wedge \\ & \quad \exists k \cdot (k \in \text{close\_up\_photo}(k) \wedge y = \text{get\_colour}(k))\}) \\ & \text{rem.op}(y \leftarrow \text{return\_colour}(x)) \end{aligned}$$

Note that, the above schema transformation partially overlaps with the refinement transformation since both remove the i_colour and e_colour attributes. We can thus restrict ourself to consider a refinement transformation t_{r1} in which the overlapping changes are missing. The applicability conditions of this restricted t_{r1} , evaluated on the new schema, can be completely discharged by reusing those checked during the previous design. This can be easily understood by noticing that t_0 only removes elements and the elements that are added in the refinement transformations are independent from the removed ones.

Let us now consider the case in which the refinement step is expanded to include additional implementational details. A precise treatment of this case requires a reasoning on the semantics of the composition operator. Due to the limitation of space, this semantics has not been included here. We then just describe this case informally.

The refinement transformation which is applied to S'_0 is the composition of t_{ri} and $t_{r_{ext}}$. It may happen that:

1. The two transformations contains the same schema modifications. For example, t_{ri} contains a $r.add.class(x, E)$ and $t_{r_{ext}}$ contains an $r.add.class(x, F)$. As the composition operator is defined only if E and F are equivalent, then it is as if the effect of the second modification be null.
2. The two transformations contains opposite schema modifications. For example, t_{ri} contains a $r.add.class(x, E)$ and $t_{r_{ext}}$ contains an $r.rem.class(x, F)$. As the composition operator is defined only if E and F are equivalent, then it is as if the effect of these two modifications is null.

3. The two transformations specify disjoint schema modifications.

In the first case, it is possible to replace t_{next} with another transformation where the overlapping changes have been removed. We can thus restrict ourself to consider only the cases 2 and 3. The situation in these cases is a little tricky since it may happen that, as effect of the composition, the proof of certain applicability conditions of t_{ri} are no longer valid. Consider, for example the case in which t_{r1} contains a $r.rem.attr(a, x, E) \circ r.rem.class(x, G)$ and t_{next} contains $r.add.attr(a, x, F)$. Imagine that a is the only attribute of the class x . If we consider t_{r1} only, x can be removed since it has no attribute defined on it. If we compose t_{r1} with t_{next} this condition is no longer true. An exact analysis of the semantics of the transformations permits to identify the situations like the ones of this example [9]. This allows to characterises completely the extent to which reuse of proofs can be exploited.

6 Reuse of Schemas

The reuse discussed above relieves from the burden of selecting the transformation to be applied for the new design. The redesign, however, is still complex, since all the intermediate schemas has to be generated. In this Section we examine the reuse of schemas to explore when the complete regeneration of the schema can be avoided.

Figure 4 shows two alternative paths for producing an intermediate schema of the new design. The schema S'_1 can be generated by following either path1 or path2. If we follow path2, we can take advantage of the already existing schema S_1 . A single transformation t_1 , applied to S_1 , suffices to return the desired schema.

Viewing the redesign as a subsequent application of path2, the redesign process takes the shape depicted in Figure 5.

In order to exploit this form of reuse, the local transformations to be applied to each of the intermediate schemas of the previous design have to be known explicitly. The theorem given below, demonstrated rigorously in [9], specifies this information.

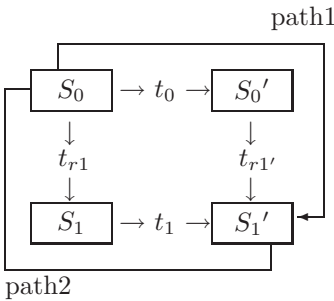


Fig. 4. Alternative paths

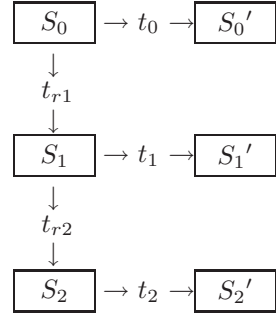


Fig. 5. The reuse of schemas

Before presenting this theorem few preliminaries are needed. First, we will assume that each transformation never add and remove the same schema component. This is not a limitation since, as we have seen before, when this occurs, the whole effect is null. Moreover, we will use the notation $[Subst]Y$ to indicate the expression that is obtained by applying the substitution $Subst$ to the expression Y . For example, $[x=E]R(x)$ is $R(E)$. Finally, we will make use of the following operator between transformations:

Definition 61 (Removal by overlapping) *Let t_i and t_j be either schema or refinement transformations. The removal from t_i of the schema transformations that overlap with those in t_j , $(t_i - t_j)$, returns the following transformation t_k :*

$$\begin{aligned}
 t_k(<Cl, Attr, IsA, Constr, Op>) = \\
 <Cl \cup (AddedCl_{t_i} - AddedCl_{t_j}) - (RemovedCl_{t_i} - RemovedCl_{t_j}), \\
 Attr \cup (AddedAttr_{t_i} - AddedAttr_{t_j}) - (RemovedAttr_{t_i} - RemovedAttr_{t_j}), \\
 IsA \cup (AddedIsA_{t_i} - AddedIsA_{t_j}) - (RemovedIsA_{t_i} - RemovedIsA_{t_j}), \\
 [RemSubst_{(t_i - t_j)}] (Constr \cup (AddedConstr_{t_i} - AddedConstr_{t_j}) - \\
 (RemovedConstr_{t_i} - RemovedConstr_{t_j})), \\
 [RemSubst_{(t_i - t_j)}] Mod_{(t_i - t_j)} (Op \cup AddedOp_{t_i} - \\
 (RemovedOp_{t_i} \cup RemovedOp_{t_j})) >
 \end{aligned}$$

where: if t_i is a schema transformation then $RemSubst_{(t_i - t_j)}$ is the empty substitution and $Mod_{(t_i - t_j)}$ is a null function. Otherwise, if t_i is a refinement transformation, then $RemSubst_{(t_i - t_j)}$ is the set of substitutions dictated by the removal of the classes and attributes effectively carried out by $(t_i - t_j)$, and $Mod_{(t_i - t_j)}$ is a function that modifies each of the schema operation specifications by adding/removing the appropriate updates for any new class or attribute added/removed by $(t_i - t_j)$.

The theorem that specifies how a correct reuse of schema can be achieved is the following.

Theorem 1 (Equivalent paths) *Let $t_{r1}; t_{r2}; \dots; t_{ri-1}$ be a sequence of refinement transformations that implement a correct refinement from the conceptual schema S_0 to the schema S_{i-1} . Let t_0 be a schema transformation that can be safely applied to S_0 and t_{ri} a refinement transformation that generates a correct refinement of the schema S_{i-1} . Let t_{i-1}/t_i be the transformation that is obtained by taking t_0 , by changing its constraint and operation parameters according to the substitutions dictated by $t_{r1}, t_{r2}, \dots, t_{ri-1}/t_{r1}, t_{r2}, \dots, t_i$, and by removing from the resulting transformation all the transformations that overlap with those in $t_{r1}, t_{r2}, \dots, t_{ri-1}/t_{r1}, t_{r2}, \dots, t_i$. Let $t_{ri'}$ the transformation that results from the removal by overlapping $(t_{ri} - t_{i-1})$ and $appl_{t_{ri'}}$ its applicability conditions. Finally, let “ $\equiv$ ” be a relation between schemas. This relation associate two schemas that differ, at most, for the set of explicit constraints. The two set of explicit constraints may have a different shape but they has to be logically equivalent. Under this conditions, the following property holds:*

$$appl_{t_{ri'}}(t_{ri}(S_{i-1})) \Rightarrow t_{ri'}(t_{i-1}(S_{i-1})) \equiv t_i(t_{ri}(S_{i-1}))$$

This theorem specifies under which conditions it is possible to replay the new design following either path1 or path2. Moreover, it describes how the transformation t_i , which specifies how the schemas the document the design process has to evolve, is derived.

As a consequence of the above theorem, the design process can be seen a sequence of steps which has the shape discussed in Section 3. The role of the transformation t_0 is played in this case by the local transformations t_i that are applied to each of the intermediate steps. This permits us to generalize the results on the reuse of the refinement steps that are given in Section 3.1.

Let us now see how the theorem above can be generalised to cover also the case in which the new refinement transformation is built as the extension of a reused one. As before, due to the limited space, we only discuss this reuse case informally. Figure 6a shows a redesign process in which a new refinement step $t_{ri'}$, which is applied to the schema S'_0 , is obtained by extending the initial refinement transformation t_{r1} with t_{rext} . Figure 6b shows a different interpretation of the same redesign process. We have said that the refinement transformations are particular schema transformations. Then t_{rext} can be interpreted as a schema transformation that is applied to S'_1 and the reused refinement transformation can be seen as applied to the schema $t_{rext}(S'_0)$. By taking this view, we reduce ourself to the situation that has been considered by Theorem 1. In this case the schema transformation is given by the sequential composition of t_1 and t_{rext} ⁴. This new transformation, as soon as the re-design proceeds, has to be submitted to the changes established by the theorem. If a new extension is met, then the schema transformation has to extended again. This behavior continues until the end of the re-design is reached.

The theorem given above has important practical consequences. In particular, it makes possible to evaluate if a change on the conceptual schema requires a change in the sequence of design steps without generating the new sequence. It also permits to generate directly the logical schema without generating the intermediate ones. This can be useful, for example, when there are strict time constraints that impose to make the system to function again as soon as possible. The sequence of schemas can be generated later, as a distinct activity useful for the successive maintenance. As far as this point, notice that, for maintenance purpose, if the design steps have not been changed, it would also be sufficient to maintain only the original design and the sequence of schema transformations that have been effectively applied to the conceptual schema. By exploiting the rule given above, the schemas of the current design can be derived.

Let us now re-examine the examples of redesign presented in Section 4 to see how, according Theorem 1, the intermediate schemas have to evolve.

Example1

This case is trivial. The transformation t_1 is obtained simply by applying to t_0 the change of variables induced by the refinement step t_{r1} :

⁴ Actually, the schema transformation is given by the sequential composition of modified versions of t_2 and t_{rext} . The modification is needed because of the difference between the composition operator “o” and the sequential operator “;”.

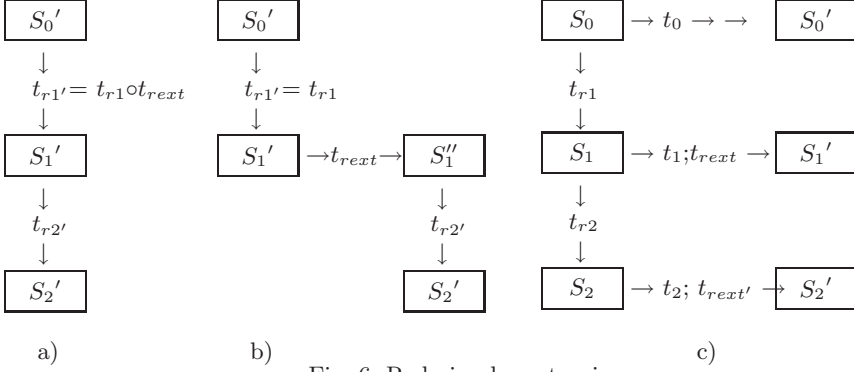


Fig. 6. Redesign by extension

$$t_1 = \text{add.constraint}(\forall m \cdot m \in (e-i \cup ei) \rightarrow \exists k \text{ long\_distance\_photo}(m) = k) \wedge \\ (\forall m \cdot m \in (i-e \cup ei) \rightarrow \exists k \text{ close\_up\_photo}(m) = k))$$

Example 2

The new schema transformation is obtained in this case by removing the overlapping transformations and by making the appropriate change of variables:

$$t_1 = \text{rem.constr}(e\_colour = \{(x, y) \cdot x \in (ei \cup e-i) \wedge \\ \exists k \in \text{long\_distance\_photo}(k) \wedge y = \text{get\_colour}(k)\}) \circ \\ \text{rem.constr}(e\_colour = \{(x, y) \cdot x \in (ei \cup i-e) \wedge \\ \exists k \in \text{close\_up\_photo}(k) \wedge y = \text{get\_colour}(k)\}) \circ \\ \text{rem.op}(y \leftarrow \text{return\_colour}(x))$$

7 Concluding Remarks

This paper has discussed the exploitation of reuse during the replaying of schema design. In particular, it has specified precisely when the refinement steps, the proofs and the schemas of the previous design can be reused. It also has made clear which modifications are needed to generate the new design.

This precise definition is a necessary step towards the construction of maintenance environments that support an automatic, or at least assisted, replay of the schema design. As the framework that has been illustrated is formalised in terms of B-Method concepts, we are exploring the possibility of building an environment on top of the tools available to discharge proofs in B.

Another on going activity concerns the exploitation of the results illustrated in other database design frameworks. We have experimented the use of the technique illustrated in carrying out the maintenance of the MIAOW system multimedia database [12]. This database, designed as part of the Marble Industry Advertising over the World ESPRIT Project (n. 3990), maintains information about stones and stone actors. The design of this database consists of a sequence

of OMT-like schemas [16]. Each schema in the sequence is generated by ad-hoc transformations. The final schema has been directly mapped into a Informix Dynamic Server schema [13]. This experience suggested improvements to the reusing technique illustrated and other interesting uses of it. In particular, we found it very useful for avoiding useless re-design. Dynamic Server permits only certain kind of modifications on the schema of an already existing database. This means that not all the modifications required at high level can be satisfied. Each time a modification to the conceptual schema was required, we first generated the last schema of the design, i.e. the schema to be mapped directly into the Dynamic Server one. This permitted us to understand immediately, without re-playing the whole design, if the change at the higher level could be supported by preserving the same design steps. This experience suggested us to explore more in deep the problem and, in particular, to extend the analysis of reuse also the the relations between the schema modifications and the actual database contents.

References

1. Unified Modeling Language,
<http://www.rational.com/uml/documentation.html> 1, 1, 5
2. Batini C., Ceri S. and Navathe S.B., *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin Cummings, 1992. 1, 5
3. Hainaut J.L., Specification Preservation in Schema Transformation - Application to semantics and statistics, *Data&Knowledge Engineering*, 16(1), Elsevier Science Publishing, 1996. 1, 1, 5
4. Ramage M. and Bennet K., Maintaining Maintainability, in *Proc. of The International Conference On Software Maintenance*, Bethesda, Maryland, November, 1998. 1
5. Abrial J.-R. *The B-Book*, Cambridge University Press, 1996. 2, 4
6. Victor R. Basili., Viewing Maintenance as Reuse-Oriented Software Development, *IEEE Software*, Vol. 7, No 1, (19-25), January 1990. 5
7. Mikael Lindvall and Kristian Sandahl., Practical Implication of Traceability, *Software Practice and Experience*, Vol.26, No.10, (1161-1180), October 1996. 5
8. Castelli D. and Locuratolo E., A Formal Notation for Conceptual Schema Specification, *Information Modelling and Knowledge Bases V*, IOS Press, (257, 274), 1994. 2
9. Castelli D. A transformational approach to database design and maintenance, *IEI-CNR Tech. Report*, 1999. 2, 8, 11, 11
10. Castelli D. and Pisani S., A Transformational Approach to Correct Schema Refinement, in *Proc. 17th International Conference on Conceptual Modeling*, LNCS, n. 1507, Springer-Verlag, Singapore, 1998. 1, 3, 7
11. Castano S. and De Antonellis V., Reuse in Object-Oriented Information Systems Development, in *Lecture Notes in Computer Science*, n. 858, (346-358), Springer-Verlag, 1995. 1, 5
12. B. Biagi and D.Castelli and F. Niccolini and S. Pisani, MIAOW: An Object Oriented Multimedia DB Application on the WWW for the Stone Market, *IEI-CNR Technical Report B4-37*, Pisa, 1996. 14

13. Informix Dynamic Server, <http://www.informix.com/informix/products/ids>. 15
14. Open Consortium, Proposing an Open standard, *Object Expert* Vol 2. No.1. 1996. 1
15. Ira D. Baxter. Design Maintenance Systems. *Communication of the ACM* Vol 35, No.4, April 1992. 5
16. Rumbaugh J., Blaha M., Premerlani W., E. Frederick and Lorensen W., *Object-Oriented Modelling and Design*, Prentice-Hall, 1991. 15

A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases

Boualem Benatallah

Queensland University of Technology
2 George Street, Brisbane GPO BOX 2434, QLD 4001, Australia
boualem@icis.qut.edu.au

Abstract. This paper addresses the design of an integrated framework for managing schema evolution. This framework is based on the adaptation and extension of two main schema evolution approaches, namely *schema modification* and *schema versioning*. The proposed framework provides an integrated environment to support different database evolution techniques (such as, modification and versions at the schema level, conversion, object versioning, and screening at the instance level). We introduce the concept of class/schema version pertinence enabling the database administrator to judge the pertinence of versions with regard to database applications. Finally, we propose a declarative language based on *OQL*, the *ODMG* query language, that the user can use to guide objects adaptation process when dealing with complex or application specific schema updates.

1 Introduction

During a database life span, the schema is likely to undergo significant changes in functional requirements (e.g., application domain change) and/or non functional requirements (e.g., performance) [10]. A schema can be updated by adding, deleting, or updating its constituents, such as attribute, method, class or inheritance relationship [12]. When a schema has been changed, objects may be inconsistent and programs may be incompatible with regard to the new schema. The support of schema evolution is the ability to allow changes of the structure and the behavior of schema's constituents without disturbing applications that are using them. This type of consistency introduces two aspects: *efficiency* and *durability*. The former relates to the fact that the mechanisms introduced, to apply the changes to schema and propagate the effects on the schema and objects, should operate efficiently in the sense that performance degradation is to be avoided. The latter issue relates to the fact that these mechanisms must ensure some degree of durability for stored information, i.e., avoid the loss of information.

1.1 Background

In this paper, we briefly overview the major approaches that are closely related to the work presented in this paper, namely *modification-based*, *versioning-based*, and *view-based* approaches [1,11].

- In a *modification-based* approach [6,9], the new schema definition replaces the old one. The *conversion* technique is used for object adaptation [6]. All existing objects must be converted to objects fitting class definitions in the new schema. The conversion may cause information loss. Another major limitation of this approach is the incompatibility of old programs w.r.t the new schema when conversion occurs.
- In a *version-based* approach [4,7,8,12], a schema update causes the derivation of new versions of classes/schema. Old programs can continue to interact with (old or new) objects in the database using the old schema, which has been retained. In this approach, two techniques of object adaptation are proposed in the literature: *screening* and *object versioning*.
 1. In the first technique [8,12], an object is associated with the class version under which the object is created. The representation of an object is never restructured after its creation. The system is responsible for ensuring access to old and new objects by programs. This technique supports program compatibility w.r.t the schema. However, it has two major limitations. The first limitation is the extra cost involved when objects are accessed. The second limitation is the information loss. The value of an added (respectively, deleted) attribute in an old (respectively, new) object is always a default value (i.e., supplied by the system), because it is impossible to restructure physically the value of an object after its creation.
 2. In the second technique, the derivation of a new version of a class, leads to the creation of a new version of each object of the class that has been versioned [4,7]. This technique resolves the problem of information loss, by supporting many object versions. However, its main limitation is the proliferation of the number of versions, due to the dynamic nature of object oriented applications. This results in a considerable overhead on the system, because the storage requirement for versions and the cost of maintaining versions relationships, increase with the number of versions.
- Some proposals (e.g., [10,11,3]) suggest the use of the *view-mechanism* to manage schema evolution. In this approach, a schema update is performed by the creation of a view that simulates the semantics of this update. The support that is provided for schema evolution by a view-based approach is not sufficient. More specifically, this approach does not provide a satisfactory solution for additive schema updates (i.e, updates requiring the reorganization of the database by information addition), such as the addition of a new attribute into the definition of a class. The information added by views (e.g, a derived attribute) are derived from other information that exist already in the database; whereas the information added by a schema update is in general new and most of the time independent from existing information.

1.2 Contributions

This paper addresses the schema update problem by integrating *modification* and *versioning* into a unified framework.

- Modification- and version-based mechanisms are complementary. A version-based mechanism can be used to overcome the problem of information loss. A modification-based mechanism can be used to control the proliferation of the number of versions by limiting the number of versions to those that are necessary.
- The existing approaches propose very primitive schema update languages. In general, these languages provide a fixed set of operations with pre-defined semantics. However, in some situations a schema update can be application specific [5]. The support of application specific updates calls for an extensible language that allows the user to specify on the fly, the desired semantic of a schema update.

Our approach can be summarized as follows:

- When a schema update is accepted, this update is effectively performed by either the modification of the schema or the generation of a new version of the schema. In the absence of the user directive, each *subtractive* update (i.e, involves information deletion) will result in the generation of new version and each *non-subtractive* update will result in a modification.
- Regarding the issue of object adaptation, our approach is based on the *evaluation of the pertinence of object versions availability*. When an object o is accessed under a version v of the schema, the system we propose checks if the object o has a version under v , i.e, the extension of the class of o (which is defined in v) contains a version of o . If not, a new version of o is generated. This new version is physically *stored* only if its availability is important (e.g., for performance improvement). Otherwise, this version is *calculated*, meaning that the version is available only during the period of use. To decide if an object version is to be stored or calculated, we have introduced the concept of *class pertinence levels*. A class may be *pertinent* or *obsolete*. It is pertinent if it is defined in the most recent version of the schema or it is used in a large number of applications. Otherwise, it is obsolete. Thus, the version of the object o , under v , is stored only if the class of o is pertinent.
- Unlike existing systems, which provide a fixed set of operations with pre-defined semantics, our approach provides an extensible language to support application specific schema updates. This language allows users to describe the relationships between the states of the database before and after the schema update. We show that, a declarative language (OQL) embedded in the primitives of our language, provides a full specification of a desired instance adaptation.

The remainder of this paper is organized as follows. In section 2, we give a brief overview of the version model that provides the basic concepts for our

approach. Section 3 addresses the issue of instance adaptation. Section 4 presents the language for specifying the semantic of a schema update. We then make some concluding remarks in section 5.

2 Basic Concepts and Definitions

The version model used in our approach supports the following concepts: *schema version*, *object version* and *version binding*. This section overviews these concepts and more details can be found in [1]. An *entity* refers to either a schema or an object of a database. At the system level, an entity is stored as a pair $(\mathbf{eid}, \mathbf{vers})$, where $\mathbf{eid}$ is the entity identifier and $\mathbf{vers}$ is the set of identifiers of its versions. The identifier of a version is also a pair $(\mathbf{eid}, \mathbf{num})$, where $\mathbf{eid}$ is the identifier of the entity and $\mathbf{num}$ is the number that the system associates with the version, at the time of its creation. The function $\mathbf{VersSet}(\mathbf{eid})$ returns the set of versions of the entity $\mathbf{eid}$.

2.1 Schema - Schema Version

A schema is associated with a set of versions which is organized in a sequence. The first version is the root version. We designate the latest version of a schema as the *current* version. A *historical* version of a schema is any version of the schema which is not the current version. We use “current” or “historical” to denote what we call the *status* of a schema version. A schema version is defined as a triple $(\mathbf{sid}, \mathbf{num}, \mathbf{val})$, where $(\mathbf{sid}, \mathbf{num})$ represents the version identifier and $\mathbf{val}$ is its value. The value of a schema version is a set of persistent classes related by aggregation and inheritance relationships.

We introduce two functions to manipulate schema versions. The function $\mathbf{SchVerClasses}(\mathbf{s}, \mathbf{n})$ returns the set of classes of the version number $\mathbf{n}$ of the schema $\mathbf{s}$. The function $\mathbf{Current}(\mathbf{s})$ returns the current version of $\mathbf{s}$. We consider one schema at a given time. Thus, a class is identified by its name and the number of the schema version. A class is then represented as a pair $(\mathbf{c}, \mathbf{n})$, where $\mathbf{c}$ is the class name and $\mathbf{n}$ is the schema version number.

2.2 Object - Object Version

An object is associated with a set of versions. Each version belongs to a different version of the schema. An object version is defined as a triple $(\mathbf{oid}, \mathbf{num}, \mathbf{val})$, where $\mathbf{oid}$ is the object identifier, $\mathbf{num}$ is the version number, and $\mathbf{val}$ is the version value. The object version is identified by the pair $(\mathbf{oid}, \mathbf{num})$. Since an object has at most one version by class, $\mathbf{num}$ is the one of the corresponding class. The value of an object version is an atomic domain element (e.g., *integer*) or a constructed domain element (e.g., a *tuple value*). An object version is an instance of a class. So, the extension of a class refers to the set of its associated object versions.

We introduce two functions to manipulate object versions. The function `Objects(c,n)` returns the set of objects associated with the class `(c,n)`. The function `Ext(c,n)` returns the set of instances of the class `(c,n)` (i.e., the extension of `(c,n)`). We also introduce three operations to manipulate object versions: **ObjectVersionCreation**, **ObjectVersionDerivation**, and **ObjectVersionDeletion**.

ObjectVersionCreation (`oid,cid`): This operation creates the first version of the object identified by `oid`. The identifier of this version is initialized by the pair `(oid,n)`, where `n` is the number of the class under which the object is created. This class is identified by `cid`. The value of this version is initialized by a default value which is conform to the type of the class `cid`.

ObjectVersionDerivation (`oid, cid1, cid2`): The classes `cid1` and `cid2` are defined in two adjacent versions of the schema. This operation creates a new version of the object identified by `oid`, as an instance of the class `cid2`. It is derived from the object version under the class `cid1`. The identifier of this version is initialized by the pair `(oid,n)`, where `n` is the number of the class `cid2`. The version value conforms to the type of the class `cid2`. It is derived from the object version value under the class `cid1`, by using the default transformation mechanism [6].

ObjectVersionDeletion (`vid`): This operation deletes the object version identified by `vid`. It triggers the update of the versions that reference the deleted version. This update replaces each reference to the (deleted) version by the default value `nil` (or by an object version whose type is a sub-type of the deleted version).

2.3 Version Binding

The management of versions is transparent to users. At the compilation time, a program/query is associated with a particular version of the schema (by default the current version). At the run time, the program refers to this schema version. The function `LinkPrSch(p)`, which binds a program to a schema version, returns the schema version under which `p` is compiled.

The reference to an object in a program, is a reference to a version of this object. It represents the object version, under a class of the schema version which is associated with the program. The function `LinkObjVer(o,p)` returns the version of the object `o` which represents `o`'s reference in `p`.

3 Combining Modification and Versioning

We assume that an update operation is applied to the current version of a schema. If the operation is accepted, then this update is performed by a schema modification or a generation of a new version of the schema. The decision to use schema modification or versioning is outside the scope of this paper. We assume that the update by modification or versioning is imposed by the user.

As mentioned in the introduction, our approach with regard to instances adaptation, combines *conversion*, *object versioning* and *screening*. To this end, the concept of *class pertinence levels* is introduced [2]. The pertinence level of a class is a means to characterize the importance of the availability of an object version, which is an instance of this class.

3.1 Class Pertinence Levels

This notion is based upon what we call a *weight* of a class. The weight of a class measures how much the availability of its instances is important for the database applications. Its definition takes into account both the number of class clients and the status of a schema version (“historical” or “current”) in which the class is defined:

- *Number of class clients*: the number of application programs referencing a class constitutes an important quantitative metric of its pertinence. When a class is deleted, the application programs which use the class, should be modified to be compatible w.r.t the schema version. So, the greather the number of programs using a class, the less is the interest in deleting it.
- *Current version of a schema*: Among all schema versions, we consider that the current version of the schema records information which may reflect more faithfully the real world.

Definition 1 (Class Weight.). *A class weight is a real value within the interval $[0,1]$ and is defined as the ratio of the number of programs associated with the class by the number of all programs associated with the schema. The classes defined in the current version of the schema have a maximal weight (i.e., 1).*

To compute the weight of a class (c,n) , we use the function $CWeight(c,n)$. Before giving the definition of this function, we introduce the following notations. We denote by S the set of schema names, C the set of class names, O the set of object identifiers, N the set of version numbers, and P the set of program names. We define:

- $Context(p)$ as the set of classes which are used in the body of the program p . This set is formed from classes used as attributes, formal parameters, and variable types.
- $Call(p)$ as the set of programs called in the body of the program p .
- $Ref(p)$ as the set of classes directly or indirectly referenced in the classes of $Context(p)$.
- $ClassesOfP(p)$ as the set of classes whose objects may be accessed by the program p . This set is defined as follows:

$$ClassesOfP(p) = Context(p) \cup Ref(p) \bigcup_{p_i \in Call(p)} ClassesOfP(p_i)$$

Given a database schema s , the function $CWeight$ is defined as follows:

$$CWeight : C \times N \longrightarrow [0, 1]$$

$$CWeight(c, n) = \begin{cases} 1 & \text{if } IsCurrentClass(c, n) \\ \frac{card\{p_i \in P / (c, n) \in ClassesOfP(p_i)\}}{AllPr(s)} & \text{otherwise} \end{cases}$$

where:

- $IsCurrentClass(c, n) = (c, n) \in SchVerClasses(Current(s))$.
- $CWeight(c, n)$ is the weight of the class (c, n) .
- $AllPr(s)$ is the number of all programs which use the database schema s .
We assume that $AllPr(s) \neq 0$.

If the value of $CWeight(c, n)$ is 1, this situation represents the fact that the availability of the instances of (c, n) is highly important, because either (c, n) is the current version or it is used in all application programs. However, the value 0 of $CWeight(c, n)$ represents a situation where the availability of the instances of (c, n) is useless, i.e., (c, n) is defined in a historical version of the schema and is not used in any application. The intermediate values represent situations where the availability of the instances of (c, n) is less or more important, depending on the perception of the database administrator (who is responsible to set up the threshold for instances availability importance).

Definition 2 (Pertinence Level of a Class.). *A class pertinence level characterizes the importance of the availability of the instances of the class with regards to database applications. Two pertinence values are considered: pertinent or obsolete. A class is pertinent if the availability of its instances is highly important. Otherwise, the class is obsolete.*

A database administrator can fix a *threshold* for the pertinence of classes. Thus, a class is pertinent if its weight is greater than a threshold (*obsolescence threshold*). Otherwise, the class is obsolete. If we denote by PL the function which determines the pertinence level of a class and OT ($OT \in [0, 1]$) the obsolescence threshold, we have:

$$PL : C \times N \longrightarrow \{“pertinent”, “obsolete”\}$$

$$PL(c, n) = \begin{cases} “pertinent” & \text{if } CWeight(c, n) > OT \\ “obsolete” & \text{otherwise} \end{cases}$$

3.2 Instance Adaptation

When an object, say o , is accessed under a class, say (c, n) , the system checks if the object o has a version under (c, n) . If not, a new version of o is generated. This version is physically stored only if (c, n) is a pertinent class. Otherwise, this version is calculated.

To generate of a new version, stored or calculated, of an object o under a class (c, n) , a sequence of basic primitives, called *object version derivation* or *deletion*, is performed on the object versions. The first primitive of the sequence is applied on a stored version of the object, called the *root of generation*.

Version Generation Primitives Five primitives are used for the generation of object versions: **OriginVersion()**, **ObjectVersionGen()**, **Nextclass()** (respectively, **Previousclass()**), and **ObjectVersionDerivation()**.

(P1) Primitive **OriginVersion()**

An object may have stored and calculated versions. During its life span, an object must have at least one stored version. The root version to generate the version of an object o under a class (c, n) is a stored version of o , whose number is the nearest to n . It is determined by using the function **OriginVersion()** which is defined as follows:

$$\begin{aligned} \text{OriginVersion} : O \times (C \times N) &\longrightarrow O \times N \\ \text{OriginVersion}(o, c, n) &\in \{ \text{ObjVers}(o) / \forall (o, j) \in \text{ObjVers}(o), |n - j| \leq |n - i| \} \end{aligned}$$

where: **ObjVers**(o) is the set of stored versions of the object o .

(P2) Primitive **ObjectVersionGen()**

This operation generates the version of an object when this version is accessed under a class whose extension does not contain a version of this object. This primitive is defined as follows:

$$\text{ObjectVersionGen}(\text{oid}, [\text{rnum}], \text{cid} [, \text{status}])$$

where: **oid** is the identifier of the object; **cid** is the identifier of the class; the optional parameter **rnum** is the number of the version from which the new version must be generated; and the optional parameter **status** takes its values in the set $\{ \text{'stored'}, \text{'calculated'} \}$.

The new version is stored (respectively, calculated) if the value of **status** is **'stored'** (respectively, **'calculated'**). If the value of **status** is not fixed, then the generated version depends on the pertinence level of its class. It is stored if its class is pertinent, otherwise it is calculated. The use of the parameter **status** is very important in some cases, especially when the generated version must be stored whatever the pertinence level of **cid**. These cases are described in the section 4.

The primitive **ObjectVersionGen()** triggers a sequence of derivation primitives on **oid**'s versions. The first primitive of the sequence is the derivation of a version of **oid** from the root version to generate the version of **oid** under the class **cid** (i.e., $(\text{oid}, \text{rnum})$ if the value of **rnum** is fixed, **OriginVersion**(**oid**, **cid**) otherwise). The last primitive of the sequence generates the version of **oid** under **cid**.

In the remainder of this section, we denote by (c_1, n_1) the class of the version **OriginVersion**(**oid**, **cid**) and (c_2, n_2) the class **cid**. There are two scenarios about the time-based relationships between the two versions:

1. If the class (c_2, n_2) is younger than the class (c_1, n_1) (i.e., $n_2 > n_1$), then in the first step, a version of the object **oid** is derived under the class of

oid, which is the successor of (c_1, n_1) in the sequence of **oid**'s classes. At the i th step, an object version is derived under the class of **oid**, which is the successor of the version of **oid**, derived at the $i-1$ th step.

2. If the class (c_2, n_2) is older than the class (c_1, n_1) (i.e, $n_2 < n_1$), then in the first step, a version of the object **oid** is derived under the class of **oid**, which is the predecessor of (c_1, n_1) in the sequence of **oid**'s classes. At the i -th step, an object version is derived under the class of **oid**, which is the predecessor of the version of **oid**, derived at the $i-1$ th step.

(P3) Primitives **NextClass()** and **PreviousClass()**

The successor (respectively, the predecessor) of a class in a sequence of the classes of an object, is determined by using the primitive **NextClass()** (respectively, **PreviousClass()**). These primitives are defined as follows:

$$NextClass : O \times (C \times N) \longrightarrow C \times N$$

$$NextClass(o, c, k) \in \{(o, i) \in Classes(o) / \forall (c, j) \in Classes(o), i > k \wedge i \leq j\}$$

$$PreviousClass : O \times (C \times N) \longrightarrow C \times N$$

$$PreviousClass(o, c, k) \in \{(o, i) \in Classes(o) / \forall (c, j) \in Classes(o), i < k \wedge i \geq j\}$$

During the generation process, stored or calculated versions of the object **oid** are generated. If the derivation of a version of **oid** under a class is required, then the pertinence level of this class is to be determined using the function **PL()**. If the class is pertinent, then a stored version of **oid** is generated. Otherwise, a calculated version of **oid** is generated.

(P4) Primitive **ObjectVersionDerivation()**

A new version of an object is derived by using the primitive **ObjectVersionDerivation()**. This primitive generates a stored version of an object, whereas here an object version may be stored or calculated. For this reason, this primitive is redefined by adding the **status** parameter, which takes its values in $\{\text{'stored'}, \text{'calculated'}\}$. Therefore, this primitive generates a stored (respectively, calculated) version if the value of **status** is **'stored'** (respectively, **'calculated'**):

$$ObjectVersionDerivation(oid, cid_1, cid_2, status)$$

Algorithm Below, we summarize the algorithm implementing the instance adaptation technique.

- When an object **o** is accessed under a class (c, n) , the system checks if the object **o** has a version under the class (c, n) . If yes, the object is used by the application without any transformation. If not, a new version of the object must be generated.

- The algorithm uses the `OriginVersion(o,c,n)` expression to determine the root version to generate the version of `o` under `(c,n)`, and uses the primitive `ObjectVersionGen(o,c,n)` to generate the version of `o` under `(c,n)`.
- After generating the version of the object `o` under the class `(c,n)`, the algorithm checks if the weight of the class of the root version is equal to `zero`, and if the object `o` has at least another stored version which differ from the root version. If yes, then the root version is deleted by using the primitive `ObjectVersionDeletion(o,r)`, such that `(o,r)=OriginVersion(o,c,n)`. The weight of a class is equal to `zero` in the following cases:
 1. The class is defined only in a hidden schema version. As pointed out before, a hidden schema version is a schema version on which a schema update is performed by a modification.
 2. The class is not defined in the current version of the schema and is not used in any application.
- The deletion of the root version means that the algorithm uses the *conversion* when it is not necessary to keep this version. So, the root version is kept only to satisfy the property requiring that an object must have at least one stored version.
- The algorithm triggers the generation of a stored object version only if the associated class is pertinent, i.e:
 - It is defined in the current version of the schema or,
 - It is used in a number of programs, judged sufficiently high by the database administrator. Consequently, if we make the simplifying assumption that objects access rate is the same for all programs, then the object versions associated with this class, are frequently accessed.

The above algorithm generates a version `(o,n)` of an object `o` from the stored version of `o`, whose number is the nearest to `n`. Therefore, the generation process triggers a minimal set of operations on `o`'s versions. Redundancy is then avoided and the time of generation is minimized.

4 Language support for Customized Schema Updates

As mentioned in the introduction, the existing approaches provide a fixed set of schema update operations with pre-defined semantics. However, such a pre-defined taxonomy of operations is not sufficient for many advanced applications where schema updates are application specific. The support of such schema updates calls for an extensible language that allows the user to specify the desired semantic of a schema update. In this way, the language provides means for the customization of schema updates to the requirement of specific applications.

We propose a language for the specification of relationships between the states of the database before and after a schema update. We show that, a declarative language (OQL) embedded in the primitives of our language, provides a full specification of a desired instance adaptation .

⁰ We note that the reader is not required to be familiar with OQL to understand the material presented in this paper, as the used OQL primitives are self-descriptive.

The aim of the proposed language is to provide constructs for the description of a relationship between two adjacent schema versions v_i and v_j ($j = i-1$ or $i+1$). This relationship specifies the desired semantic of the schema update that generates the version v_j from v_i . The information provided by these relationships is used during instances adaptation. A relationship expresses a mapping between the respective instances of v_i and v_j . The source of a mapping is a set of classes defined in v_i and the target of this mapping is a class defined in v_j . In the remainder of this paper, we call this relationship an *Instance Mapping Descriptor (IMD)*.

Intuitively, to describe an IMD, we must localize *source objects* that are related to a given *target object* and specify how these objects are related. Thus, in our approach an IMD consists of two basic expressions: the *localization expression* and the *dependency expression*. The next subsection presents the instance mapping language through an example. Subsection 4.2 defines an instance mapping descriptor.

4.1 An Example

To illustrate the process of an IMD specification, we consider the database schema that contains the class **CProgram**, whose objects are programs written in the C language and the class **JavaProgram**, whose objects are programs written in the Java language. Let us consider the schema update that merges the classes **CProgram** and **JavaProgram** into one single class, called **Program**. In this example, we want that the definition of the new class **Program** will contain an attribute, called **Language**, whose value is a **string** indicating the language used to write the corresponding program. The definition of the other attributes of the class **Program** is ignored here for clarity reasons.

We will now describe intuitively our instance mapping language. When updating the schema, the user can use this language to specify a mapping between the instances of the database before and after the schema update. In this example, we want to specify the following facts:

- All objects of **CProgram** and **JavaProgram** are also objects of **Program**.
- The value of the attribute **Language** in a version of an object of **Program** which is also an object of **CProgram** (respectively, **JavaProgram**) is ‘‘CProgram’’ (respectively, ‘‘JavaProgram’’).

As pointed out before, our language features two expressions for specifying an instance mapping: the localization expression and the dependency expression. The first expression localizes source objects that have a link with a given target object. The second expression describes the relationship between the value of the version of an object under the target class and the values of versions of its related objects under the source classes.

In this example, the instance mapping can be specified by the following descriptor:

source {CProgram,JavaProgram} target Program
localization query Objects(CProgram) union Objects(JavaProgram) link <i>same-as-source</i>
dependency Language derived with element(select * from {'CProgram','JavaProgram'} as name where name in ClassNames(this))

Where:

- `ClassNames(o)` is a set that contains the names of the classes of the object `o`.

The target is `Program` and the source is the set `{CProgram,JavaProgram}`. The *localization clause* introduces the localization of `CProgram` and `JavaProgram` objects. The *dependency clause* introduces the relationship between the version of an object under `Program` and the version of this object under `Cprogram` or `JavaProgram`.

Let us now show how an IMD is used to customize the semantics of a schema update. In our approach an IMD is used to:

1. *Initialize the extension of the target class.* After merging `CProgram` and `JavaProgram` into `Program`, the system uses the above descriptor to initialize the extension of `Program` as follows :
 - For each object in `Objects(CProgram) union Objects(JavaProgram)`, a new version is generated under the class `Program`.
 - The value of the attribute `Language` in the new version (generated in the previous step), is initialized to be the **name of the class** of the source object. A new version of an object `o` of the class `c` (i.e, `CProgram` or `JavaProgram`), under the class `Program`, is derived by using the following primitive:

ObjectVersionDerivation(c,Program, o)

Language derived with `element(select *
from {'CProgram','JavaProgram'} as name
where name in ClassNames(o))`

2. *Propagate the update of the extension of a source class to the extension of a target class.* When a new version of an object `o` is created under the class `CProgram` or the class `JavaProgram`, another version of `o` is created under the class `Program`. The last version is derived from the first one using the primitive described above.

In the remainder of this paper, we focus on the first issue. Details about the second issue can be found in [1].

4.2 Instance mapping descriptor

The syntax we use to specify an IMD is the following:

source { <i>class-name</i> +} target <i>class-name</i>
localization query <i>localization-expression</i> link <i>target-object-id</i>
dependency (<i>attribute-name attribute-specification</i>)+ <i>attribute-specification</i> ::= derived <i>attribute</i> * with <i>function-body</i> <i>target-object-id</i> ::= <i>same-as-source</i> or <i>new</i> <i>attribute</i> ::= <i>attribute-name class-name</i> <i>attribute-name, class-name, function-body</i> ::= string

The *source* and *target* clauses are self-descriptive. The following subsections describe the *localization* and *dependency* clauses.

Localization clause. The localization expression consists of two elements. The first element is a declarative query that selects the identifiers of source objects that will be related within an identifier of a target object. This query is introduced by the keyword **query**. The scope of this query contains objects of source classes. The second element is the expression that defines how a target object is linked with a set of source objects. This expression is introduced by the keyword **link**. Without loss of generality, we consider two types of links:

1. **The identifier of the target object is new.** In this case, the keyword **link** is followed by the keyword *new*. For example, assume that the schema *s* has two versions (*s*,0) and (*s*,1). The version (*s*,0) contains the classes *Student* and *Employee*. The version (*s*,1) contains the class *Employee-Student*. Consider the schema update that *merges* the classes *Employee* and *Student* into the class *Employee-Student*. Assume that we want to implement this schema update as follows: the objects of *Employee-Student* are constructed by joining pairs of objects from *Employee* and *Student* on the attribute *Name* (we assume that *Name* is a key attribute for *Employee* and *Student*). To this end, we specify the following descriptor:

source { <i>Employee, Student</i> } target <i>Employee-Student</i>
localization query select struct(id1 : x, id2 : y) from Objects(<i>Employee</i>) as x, Objects(<i>Students</i>) as y where x.Name=y.Name link <i>new</i>
...

When the class **Employee-Student** is created, its extension is initialized by using the information provided in the previous IMD. First, the query of the localization expression is evaluated. This query returns a set of pairs of objects formed by joining **Objects(Employee)** and **Objects(Student)**. Two objects form a pair if they have the same name. Second, the link expression of IMD informs the system that for each pair (x,y) of the set returned by the query, a new object o of the class **Employee-Student** is to be created. The link between (x,y) and o is materialized for future use. We use the expression **TargetLink(o)** to determine the set of source objects that are related with the target object o.

2. **The identifier of the target object is a source object.** In this case, the keyword **link** is followed by the keyword *same-as-source*. Consider the schema update that *merges* the classes **CProgram** and **JavaProgram** into the class **Program**. When the class **Program** is created, its extension is initialized as follows. The query **Objects(CProgram) union Objects(JavaProgram)** is evaluated. After that, each object of the returned set is added to the extension of the class **Program**. This means that for each of these objects, a new version is to be created.

Dependency clause. The dependency expression of an IMD specifies the relationships between the attributes of the target class and those of the source classes. It is introduced by the *dependency clause*. For a given attribute, this expression defines how to compute the value of this attribute. Thus, it provides a means to initialize or modify the value of the version of an object under the target class.

In an IMD, the specification of the dependency expression of an attribute contains two elements. The first element is the name of the attribute of the target class. The second element is the function that computes the value of this attribute. This function is introduced by the keyword **derived with** and is an *OQL* query. Consider the schema update that *merges* the classes **CProgram** and **JavaProgram** into the class **Program**. The dependency expression of the attribute **Language** is described as follows:

```
Language derived with
  element(select *
    from {'CProgram','JavaProgram'} as name
    where name in ClassNames(this))
```

When a new version of an object in **Objects(CProgram) union Objects(JavaProgram)** is created under the class **Program**, the value of the attribute **Language** in this version is initialized using the information provided by the above dependency expression. That is, the **name** of the class of the root version, is set to the value of the attribute **Language**.

5 Conclusion

In this paper, we proposed a unified framework for supporting dynamic schema evolution in object oriented databases. A schema update is performed by a modification of the schema, only if it is not subtractive (i.e, does not involve information deletion) or the user imposes the “modification” mode. The proposed technique for supporting instances adaptation after a schema update, is based upon the evaluation of the pertinence of the availability of object versions w.r.t applications. The number of versions is limited to those which are frequently accessed. We proposed an extensible language that can be used to specify a desired semantics of a schema update. This language provides means for the customization of schema updates to the requirement of specific applications. We have also implemented the proposed schema update mechanism on top of an object-oriented DBMS. Due to space reasons, the description of the prototype is outside the scope of this paper.

References

1. Benatallah, B. *Evolution du schema d'une base de donnees a objets: une approche par compromis*. PhD dissertation, University of Joseph Fourier, Grenoble, March 1996. 16, 19, 27
2. Benatallah, B. and Tari, Z. *Dealing with Version Pertinence to Design an Efficient Object Database Schema Evolution Mechanism*. The IEEE Int. Database Engineering and Applications Symposium - IDEAS '98, July 1998, Cardiff, Wales, UK. 21
3. Breche, P. and Ferrandina, F. and Kuklok, M. *Simulation of schema and database modification using views*. Proc. of DEXA'95, London, UK, 1995. 17
4. Clamen, S. *Schema Evolution and Integration*. Distributed and Parallel Databases Journal, Vol. 2(1), Jan. 1993. 17, 17
5. Claypool, K. and Rundensteiner, E. *Flexible Database Transformations: The SERF Approach*. IEEE Data Engineering Bulletin 22(1), 1999. 18
6. Ferrandina, F., Meyer, T., and Zicari, R. *Schema and Database Evolution in the O2 system*. Proc. of the 21th VLDB Int. Conf., Zurich, Sept. 1995. 17, 17, 20
7. Fontana, E. Dennebouy, Y. *Schema Evolution by using Timestamped Versions and Lazy Strategy*. Proc. of the French Database Conf. (BDA), Nancy, Aug. 1994. 17, 17
8. Monk, S. and Sommerville, I. *Schema Evolution in OODBs Using Class Versioning*. SIGMOD RECORD, 22(3), Sept. 1993. 17, 17
9. Penny, D. and Stein, J. *Class Modification in the GemStone Object-Oriented DBMS*. Proc. of the ACM OOPSLA Int. Conf., Sept. 1987. 17
10. Ra, Y. and Rundensteiner, E. *A Transparent Schema-Evolution System Based on Object-Oriented View Technology*. TKDE, 1997. 16, 17
11. Rodick, J. *A survey of schema versioning issues for database systems*. Information and Software Technology, 1995, 37(7) 383-393, Elsevier Science B.V. 16, 17
12. Zdonik, S. *Object-Oriented type evolution*. Advances in Database Programming Languages. ACM Press, (Bancilhon F., Bunema P. editors), 1990, pp. 277-288. 16, 17, 17

An Incremental and Semi-automatic Method Inheritance Graph Hierarchy Construction

Mohamed M. Gammoudi¹, Ibtissem Nafkha¹, Zair Abdelouahab²

¹ University of Tunis, Department of Computer Science,
Campus Universitaire, Le Belvédère, 1060, Tunis, Tunisia.

E-mail: Mohamed.gammoudi@fst.rnu.tn Phone/Fax: (+216)1 716691.

² Universidade Federal do Maranhao, Campus do Bacanga,
65080-040 São Luis –MA, Brasil.

E-mail: Zair@elo.com.br Phone: (+55)982353289.

Abstract In [16] we proposed a semi automatic method for generating inheritance graph hierarchy to assist the designer during the definition of a conceptual schema. The motivation of this work was the very few approaches which attempt to provide methods and tools for designing inheritance graph in object databases [30], and object software [1]. To address some limitations found in our work, we propose a new method whose principles are : (i) From a binary relation which represents the links between entities and their properties and methods, our proposed heuristic generates and organizes incrementally a set of *Optimal rectangles* into a *Brute Inheritance Graph* (BIG). (ii) BIG is refined and submitted to the designer. (iii) The designer can modify, add or remove classes, attributes or methods in the binary relation and activates steps (i) until he obtains a proper class hierarchy or an *Optimal Inheritance Graph*.

Keywords: Conceptual Schema, Class, Inheritance Graph Hierarchy, Binary Relation, Incremental and Rectangular Decomposition, Optimal Rectangle.

1. Introduction

Inheritance is a powerful mechanism in Object Oriented Programming. This mechanism supports the class hierarchy design and capture the *Is-A* relationship between a superclass and its subclass. It is also a promising programming tool with good properties, to reduce the cost of software development and to enhance software reuse and object-oriented design. This property has been widely applied in object oriented software and object oriented databases [27].

There are two kinds of inheritance:

- *Simple inheritance*, where a class can have only one direct superclass. In this kind a class inherits attributes and methods from only one class.
- *Multiple inheritance*, or *repeated inheritance* [12] where a class inherits attributes and methods from more than one direct superclass.

In systems which support simple inheritance, classes form a class hierarchy. However, systems which support multiple inheritance, the classes form a *Rooted Directed Graph*, sometimes called a *Class Lattice* [12]. In fact, several research efforts use *Galois Lattice* structure as a support to define class hierarchy [30] or for interface hierarchy [19].

When we develop an object oriented system, one we must define classes, methods and all relationships between classes such as: aggregation (Part-of),

Specialisation / Generalisation Is-A, and so on. It is very important that the designer is provided with tools to help him with his task, such as a tool for generating and displaying the inheritance graph, to allow the designer to efficiently control the conceptual schema such as name-conflicting, redundant inheritance, redundant Is-A relationship and cyclic inheritance [10].

In our approach, we introduce an incremental and systematic method for the construction of inheritance graph and the feedback of the designer. In fact, from a binary relation, we select and organize incrementally a set of optimal rectangles into a *Brute Inheritance Graph* (BIG). After removing the redundancy from the BIG, the system interacts with the designer to validate the graph and generates the *Optimal Inheritance Graph*.

This paper is organized as follows: section 2 discusses related work and presents an outline of our approach. In section 3, we present mathematical foundation on which our method is based and which are necessary for its understanding. Section 4, introduces an incremental and Optimal Inheritance Graph generation method; then, we compare it with some other methods. Section 5 concludes the paper by illustrating the advantages and limitations of our method and discussing our prospects for future work.

2. Related Work

The design and modeling of class hierarchies is the subject of much research nowadays. Some approaches are based on conceptual clustering methods and taxonomic reasoning in knowledge representation system, such as the approach introduced by Step and Michalski in [25]. This approach is considered as an adequate solution to the limitation of conventional techniques which are used to construct classification of a given set of objects. and are based on cluster analysis and numerical taxonomy. Conventional techniques are inadequate because they organise objects into classes on the basis of a numerical measure [3]. Some other research efforts use structural elements to generate inheritance hierarchies such as the structure of Galois Lattice. This structure is used to represent the inheritance hierarchy. It is understandable, because multiple inheritance is defined as a class lattice [11]. Galois Lattice is also used in several domains, such as in interface class hierarchy [20], [21], knowledge representation and learning theory[29] and in Class Hierarchy designing [30].

These methods are efficient and provide good results. However, they use a Galois Lattice structure which has a high number of classes compared to the initial number of given objects [30]. This high number of classes in Galois Lattice implies a lot of redundancy. To remove this redundancy, a lot of processing is required. As an alternative to this limitation, we introduce a method which has a simple mathematical foundation and provides a number of nodes less than those generated by methods based on Galois Lattice structure. This is ensured by the use of the *Optimal Rectangle* notion. To generate a proper Inheritance Graph Hierarchy we proceed as follows:

- First, we use an incremental algorithm to decompose a binary relation into a set of *optimal rectangles*, which constitutes its minimal cover. This decomposition is an approximate solution of NP-Complete problem; at the same time, we organise them

incrementally into a *Brute Inheritance Graph* (Acyclic) using the partial order relation;

- Second, the *Brute Inheritance Graph* is simplified by removing redundant entities or properties;
- Third, the system uses some inferences rules to analyse the BIG and shows it to the designer to be validated. The feedback of the designer allows us to obtain an *Optimal Inheritance Graph* avoiding problems like name-conflicting, redundant inheritance, and so on.

3. Mathematical Foundations

A binary relation R on set E is defined as a subset of the cartesian product $E \times E$. We denote by eRe' the fact that an argument e (or input) of E is linked with an image e' (or output) of E by R . Among the relations on a set E , we can mention the *identity relation* I and the *null relation* $\emptyset$ (we shall use this symbol to denote the null set $\emptyset$). If S is a subset of E , then we denote by $I(S) = \{(e,e) \mid e \in S\}$, the image set of S by I .

We can associate the following subsets of E with a given binary relation R :

- the *image* set of e is defined by : $e. R = \{e' \in E \mid e R e'\}$;
 the *antecedents* of e' are defined by : $R. e' = \{e \in E \mid e R e'\}$;
 the *domain* of R is defined by : $\text{Dom}(R) = \{e \mid \exists e' \in E : e R e'\}$;
 the *range* of R is defined by : $\text{Range}(R) = \{e' \mid \exists e \in E : e R e'\}$.

3.1 Rectangle

Let R be a binary relation defined on sets E and F . A rectangle of R is a couple of sets (A,B) such that $A \times B \subseteq R$. A is the domain of the rectangle and B is its Range (see figure 3.1).

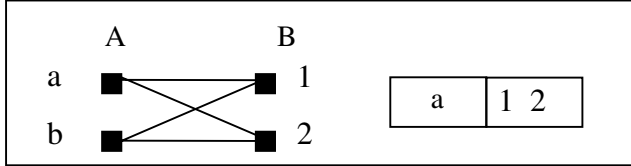


Figure 3.1 : An example of two representations of a rectangle (A,B)

Remarks:

The correspondence between rectangles (A_i, B_i) and the associated rectangular relations [6] $A_i \times B_i$ is a bijective one, except when $A_i = \emptyset$ or $B_i = \emptyset$. For instance, the rectangles $(\emptyset, B_1)$ and $(A_1, \emptyset)$ are both associated with the null rectangular relation $\emptyset$. The main reason for making a distinction between rectangles and rectangular relations is that the concept of a rectangle enables us to obtain a lattice structure.

3.2 Maximal Rectangle

Let (A,B) be a rectangle of a given relation R defined on S . The rectangle (A,B) is said to be *maximal* if whenever $A \times B \subseteq A' \times B' \subseteq R$, then $A = A'$ and $B = B'$.

Remark : The notion of maximal rectangle is not new, it is found with different names such as: Complete couple in [19], or Concept in [29].

3.3 Gain

With respect to the economy of concept, the *gain* which is obtained by a given rectangle

$RE = (A, B)$ (or the associated rectangular relation $A \times B$) (see figure 3.2) is computed by the following function: $g(RE) = (\|A\| * \|B\|) - (\|A\| + \|B\|)$ where $\|A\|$ denotes the cardinality of the set A and $\|B\|$ the cardinality of the set B .

Remark : The main advantage of RE's is that they reduce the number of concepts which is an important contribution and the approach yields even better maximizing the economy of concept as the number of entities or properties increases.

3.4 Optimal Rectangle

A rectangle $RE = (A, B)$ which contains an element (a, b) of a binary relation R is said to be *optimal* if it realizes the maximal gain $g(RE)$ among all the maximal rectangles (see section 3.3) which contain (a, b) . Figure 3.2-a represents an example of a relation R . The figures 3.2-b, 3.2-c, and 3.2-d represent three maximal rectangles of R which contain the couple $(y, 3)$. The gains obtained, in the sense of memory space with these three maximal rectangles are respectively 1, 0, and -1. Then, the optimal rectangle which contains the pair $(y, 3)$ of R is the rectangle illustrated by figure 3.2-b because it allows to achieve the maximal gain.

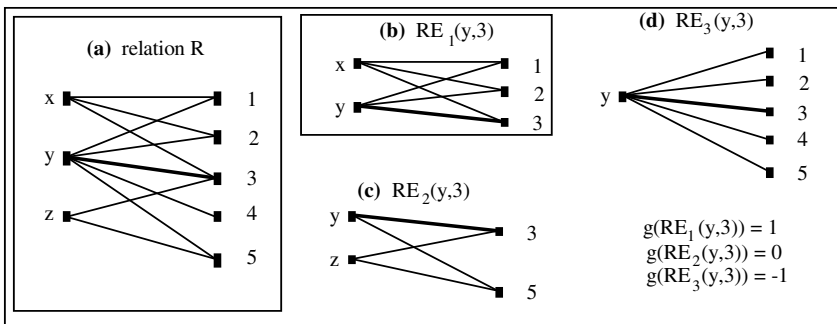


Figure 3.2 : Illustration of Optimal Rectangle and Gain Notions

3.5 The infimum of two rectangles

Let (A, B) , (A', B') , (A'', B'') be three rectangles of a given relation R . The rectangle (A, B) is said to be *infimum* of (A', B') , (A'', B'') if $A \subseteq A' \cap A''$ and $B \subseteq B' \cap B''$.

The figure 3.3 represents the rectangle $R1 = (\{a, b\}, \{1, 2\})$ which is an infimum rectangle of $R2 = (\{a, b, c\}, \{1\})$ and $R3 = (\{a, b, d\}, \{2\})$.

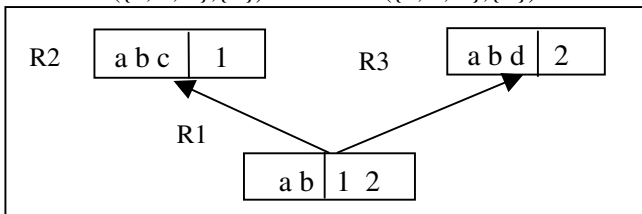


Figure 3.3 : Illustration of an infimum of two rectangles

3.6 Minimal cover

Let R be a binary relation. A cover of R is a set of rectangles $C = \{RE_1, RE_2, \dots, RE_n\}$ such that for each element (x,y) of R there exists $RE_i \in C$ which $(x, y) \in RE_i$. A minimal cover of R is the smallest subset of C which covers it.

3.7 Partial order relation

Proposition 1:

The relation defined below on the set of optimal rectangles of a binary relation R is a partial order relation : $\forall (A_1, B_1)$ and (A_2, B_2) two optimal rectangles of R , $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ and $B_2 \subseteq B_1$. The proof of this proposition can be found in [13].

3.8 Optimal rectangular graph

Let R be a binary relation defined on the sets E, F . Let G be a set of optimal rectangles which cover R . $(G, \leq)$ forms a graph. This graph has a supremum $(E, \emptyset)$ and an infimum $(\emptyset, F)$.

Remark: Let G' be the set of maximal rectangles that are subset of some relations R_i , then $(G', \leq)$ forms a Galois lattice having as a supremum $(E, \emptyset)$ and as an infimum $(\emptyset, F)$ [5].

3.9 Galois Lattice

Let R be a finite binary relation defined on E and F there is a unique Galois Lattice corresponding to R [17]. Each element of the lattice must be a maximal rectangle as defined in 3.2, noted (X,Y) , composed of a set $X \in P(E)$ and a set $Y \in P(F)$. Let f and g defined as following :

$$f = \{ (X,Y) / \forall x \in X, \forall y \in Y (x, y) \in R \} ; g = f^{-1} = \{ (Y, X) / (X,Y) \in f \}$$

where the couple of functions (f, g) is said to be a Galois Connection between $P(E)$ and $P(F)$ and the Galois lattice $(GL, \leq)$ for the binary relation is the set of all maximal rectangles with the partial order relation. The partial Order is used to generate the graph of the Galois lattice which is called Hasse Diagram. Galois lattice has a supremum $(E, \emptyset)$ and an infimum $(\emptyset, F)$, for more details on Lattice Galois the reader should consult [3].

4. Incremental rectangular decomposition of a binary relation

In this section, we present the principles of our method and we show how we apply it to generate the Brute Inheritance Graph. Let R be a finite binary relation. We mean by Incremental Rectangular Decomposition of R , the selection of a set of optimal rectangles which form a minimal cover of R . A binary relation R , from E to F , defines edges of a bipartite graph on $E \cup F$. Then, a rectangle is a complete bipartite sub-graph of the graph $(E \cup F, R)$. The problem of optimal rectangle selection containing an element (a, b) , from a binary relation R can be seen as the choice of a subset of maximal rectangles which are selected. This problem is similar to a selection of a complete bipartite sub-graph with maximal cardinality and which is contained in bipartite graph. This problem has been the subject of much research in the part [8, 18]. The proposed algorithms in [4] give an exact solution to the lattice construction problem, but they are combinatorial, and are not suitable if the cardinality of a finite

binary relation is large (more than 16 couples). Furthermore, we distinguish two approaches to the construction of Galois Lattice Construction. The first one includes, [17, 22, 24] and focuses on how to select the nodes of the Galois Lattice, and the second current which includes [8, 20, 21], introduces incremental algorithms to construct Galois Lattice. The problem is that the number of Galois Lattice nodes is large. As an alternative, we use the notion of optimal rectangle to optimize the number of nodes without loss of information.

The incremental aspect of our proposed method permits to suppress, add or modify a node of a graph, avoiding the decomposition in each operation, as in the case of RDBR [16].

In this section, we present the key issues of our incremental algorithm for the decomposition of a binary relation, that We call INC-RDBR.

4.1 Incremental generation of optimal inheritance graph

As shown in the figure 4.1, in this section we treat the following points :

- (1) how we generate and organize incrementally the set of optimal rectangle into a BIG;
- (2) how the BIG is simplified and submitted to the designer;
- (3) how the system interacts with the designer to refine BIG and to define a *Proper Inheritance Graph* or an Optimal Inheritance Graph (see figure 4.1).

To illustrate the algorithm, we are going to show how we generate the class hierarchy of the example taken from [16, 30] and presented in figure 4.2.

4.1.1 Generation of the Brute Inheritance Graph (BIG) The inheritance graph corresponding to this binary relation is generated in an incremental manner ; that is through successive calls to INC-RDBR.

Let E be the set of entities in the real world, F the of properties corresponding to entities E and R of the binary relation from E to F . Let f be a function as it is defined in (3.10). Let G be the graph corresponding to R . Let x^* be the element to add to the graph G and R^* , E^* , F^* , G^* the respective versions of R , E , F , G after adding x^* . A Node of G^* is represented by a couple (X, X') and a node of G is represented by (Y, Y') . The graph G^* may have four types of nodes which we classify as follows :

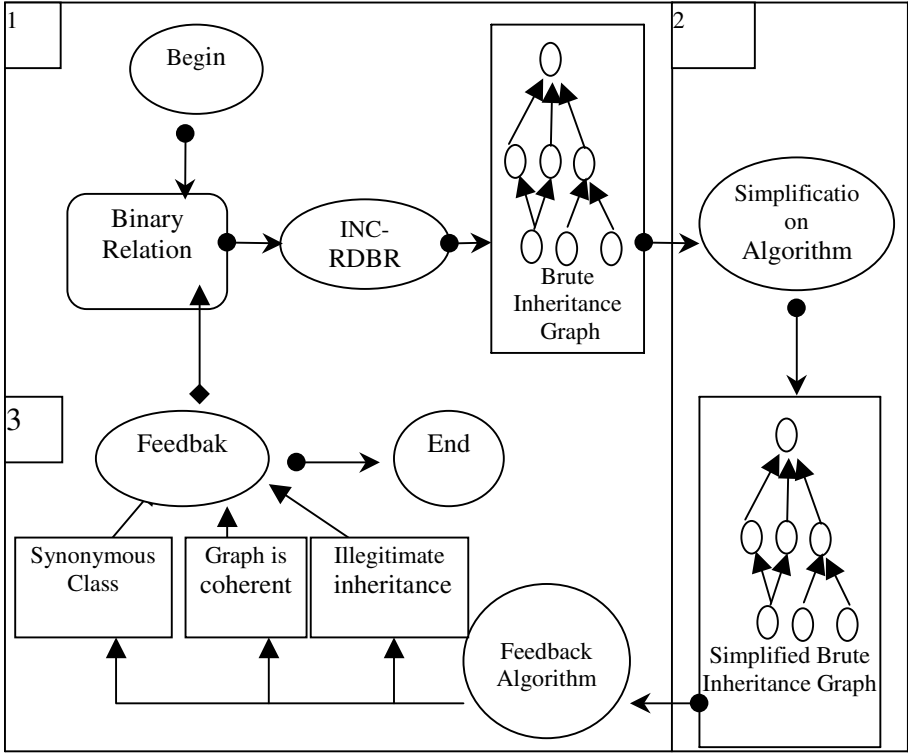


Figure 4.1: Steps of Incremental and Optimal Inheritance Graph Generation

1. **New Node** : is a node where X' does not exist in G (and cannot be inferred from the parents nodes (fathers)).
2. **Old node** : is a node in which X et X' are not modified.
3. **Modified node** : is a node where $X' \in G$ but X is modified in G^* .
4. **Suppressed Node** : is a node where (X, X') can be deduced from the parents nodes.

The set X of a modified node is always equal to $Y \cup \{x^*\}$ where (Y, Y') is in G and $X'=Y'$. The node (Y, Y') such that $Y' \subseteq f(x^*)$ produces a modified node in G^* .

The creation of a new node is more complicated than its modification. Each new set X' in G^* is the result of intersecting $f(x^*)$ with the set in G and respecting the notion of non infimum rectangle (see section 3.5). The latter requires that the node (X, X') of G^* should correspond to an optimal rectangle. This means that the node (X, X') should not be deduced from the parents nodes. In addition, the node $(x^*, f(x^*))$ is added to the graph G^* if $f(x^*)$ is not included in F . The following algorithm INC-RDBR is :

Algorithm INC-RDBR ($G, (x^*, f(x^*)), R, G^*, R^*$)

Begin

/* R is defined on $E \rightarrow F^*$ */

1. $E^* := E \cup \{x^*\}$;

/* State Initialization of graph's nodes */

2. For All node N of G **do**

3. N.Flag := False ;

4. Next node;

5. End For;

6. If ($f(x^*) \not\subset F$) **Then**

/* Add a new node with N' as its name, $\{x^*\}$ its domain, $f(x^*)$ its range, its flag is equal true. This insertion is done at the level which has cardinality equal $\|f(x^*)\|$ */

7. Add_node ($\|f(x^*)\|$, N', $\{x^*\}$, $f(x^*)$, true) ;

8. $F^* := F \cup f(x^*)$;

9. End If;

10. For all Level_N of graph G **do**

11. For all node N of Level_N **do**

12. If (N.Flag = False) **then**

13. **If** $f(x^*) = N.Range$ **Then** /* Modified node */

14. N.domain = $N.domain \cup \{x^*\}$;

15. N.Flag := true ;

/* This modification is propagated on list of fathers of node N of the level Level_N */

16. Propagation_modification (Level_N, list_father(N))

16. Exit ;

17. **Else** /* Int is the variable contained the intersection between $f(x^*)$ and N.range */

18. $Int := f(x^*) \cap N.range$

19. **If** $Int = f(x^*)$ **Then** Exit **End If;**

20. **If** ($Int = N.range$) **Then** /*modified node*/

21. N.domain := $N.domain \cup \{x^*\}$;

22. N.flag := true ; /* This modification is propagated on list of fathers (parents) of node N of the level Level_N */

23. Propagation_modification(Level_N, list_father(N));

24. **Else** /* level_Int is the level correspond to $\|Int\|$ */

25. **If** ($Int \not\subset [\cup ((list\_father(N)).range)]$ **or** ($level\_Int > HL$ and $(N.domain \cup \{x^*\}) = E^*$) **Then** /* Add a new node with N' is its name, its domain is $N.domain \cup \{x^*\}$, its range is Int, flag is equal true at level which has $\|Int\|$ as cardinality */

26. Add_node ($\|Int\|$, N', $N.domain \cup \{x^*\}$, Int, true) ;

27. Update_list_father(N', N) ; /* add N' to list of fathers of N at queue */

28. Update_list_children(N, N') ; /* add N to list of children of N' at queue
simplification of infimum rectangles in the graph G */
infimum_rectangle_Simplification () ;

29. End If

30. End If

31. End If

32. End If

33. Next Node;

34. End For

35. Next level;

36. **End For**

37. $HL := High\_Level(G)$; /* this procedure returns the high level of G */

38. **End**

Procedure infimum_rectangle_Simplification ()

Begin

For all level_N of graph G **do**

For all node N of level_N **Do**

If $(N.range \subseteq [\cup ([list\_father(N)].range)] \text{ or } (N.domain \neq E \text{ and level\_N} = High\_level))$ **Then**

Delete N ;

/* To reorganize links between fathers of N and its children */

For all node N' of list_father (N) **do**

For all node N'' of list_children(N) **do**

Delete N from N''. list_father (N'') ;

Delete N from N' . list_father (N') ;

If $(N''.list\_father(N'') = NULL)$ **then**

Link (N', N'') ; /* add an edge between N' and N'' in the graph */

Update_list_father (N' , N'') ; /* add N' to list_father of N'' at queue */

Update_list_children (N'', N') ; /* add N'' to list_children of N' at queue

*/

End If;

End For

End For

End If

Next node ;

End For

Next level ;

End For

End.

Procedure propagation_modification (first_level, list_father(N))

Begin

/*first_level: The level from where the propagation is done on the nodes of father of N*/

For each father N' of N **do**

If N'.flag = false **then**

For level_N' := (first_level +1) to High_level **do**

N.domain := $N.domain \cup \{x^*\}$;

N.flag = true;

/* This modification is propagated on list of father of N' */

Propagation_modification(level_N' , N');

End For;

End If;

Next father node of N;

End for;

End;

INC-RDBR has as an input a graph G , a relation R , and a given line from R to treat. As an output it has a graph G^* and a relation R^* . We start with a given relation and an empty graph and we go along the lines of the binary relation. R and G are modified accordingly.

Initially, $R=G=\emptyset$; and starting by the first line to treat $L_1 = \{\{a1\}, \{b1, b3\}\}$. $f(x^*)$ is equal to $\{b1, b3\}$; it is not a subset of F which is considered as empty then the statement 7 is executed, a new node $N' = (\{a1\}, \{b1, b3\})$ is created and F is incremented by $f(x^*)$.

Entities \ Properties		Name	Credits	Salary	Vacations	Type_of_Trainin	Penalty	Type_of_Viol
		b1	b2	b3	b4	b5	b6	b7
Staff	a1	1	0	1	0	0	0	0
Student	a2	1	1	0	0	0	0	0
Vacations_Student	a3	1	1	1	1	0	0	0
Training_Staff	a4	1	1	1	0	1	0	0
Bad_Staff	a5	1	0	1	0	0	0	1
Bad_Student	a6	1	1	0	0	0	1	1

Figure 4.2: Binary matrix representing the relationship R between Entities and their properties.

The higher level of G is equal to zero and then none of the instructions is executed and thus terminating the treatment of L_1 . The graph G^* contains the only created node :

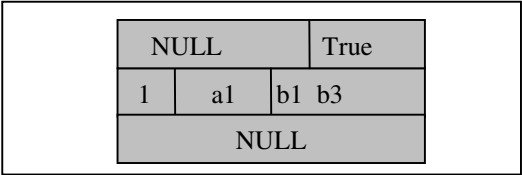


Figure 4.3 : Brute Inheritance Graph G_1

For the second line $L_2 = \{\{a2\}, \{b1, b2\}\}$ of the binary relation: the algorithm INC_DRBR is executed with the arguments G and R as its inputs that are respectively the graph (G_1) and the resulting relation from the treatment of L_1 . Since $f(x^*)$ is not part of F , the instruction 7 gives a new node $N' = (\{a2\}, \{b1, b2\})$. We start traversing the nodes of the graph that have a superior level which is equal to 1. One

node is found ($\{a1\}, \{b1, b3\}$), the condition of the instruction 18 is verified and gives rise to the intersection of $f(x^*)$ with its range a value $\{b1\}$. This intersection verifies the instruction 26 : $j=2$ et $X' \cup \{x^*\} = \{a1, a2\} = E^*$ and thus a new node $N' = (\{a1, a2\}, \{b1\})$ is created with its links. The resulting graph is shown in the following figure:

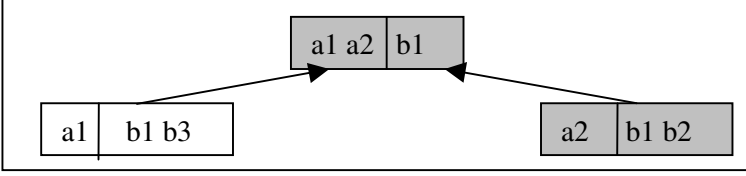


Figure 4.4 : Brute Inheritance Graph G_2

Taking as a base the graph G_2 , the treatment of the third line $L_3 = \{ \{a3\}, \{b1, b2, b3, b4\} \}$ of the relation is started with the test on the inclusion of $f(x^*)$ in F ; this condition is not verified, F is then modified by adding this set $f(x^*)$ and a new node $N' = (\{a3\}, \{b1, b2, b3, b4\})$ is created. Since the superior level is equal to two, we should treat its nodes. The first encountered is $(\{a2\}, \{b1, b2\})$; it verifies the condition 18 and gives a value to Int equal $\{b1, b2\}$ which satisfies the condition 21. The node is then modified, marked and the procedure that propagate the modifications is started. The modification consists of adding $\{x^*\}$ to the domain of the node and concerns as well all its parents. In our example, the only parent node (father) is $(\{a1, a2\}, \{b1\})$ which should be modified and marked. The treatment of the following node $(\{a1\}, \{b1, b3\})$ and gives an intersection equal to $\{b1, b3\}$ which is equal to its range ; this is a modified node which should propagate the modifications to its fathers. The treated node has only one parent (father) which is already marked. The set of nodes of the graph are all treated and thus the resulting graph after treating L_3 is as follows :

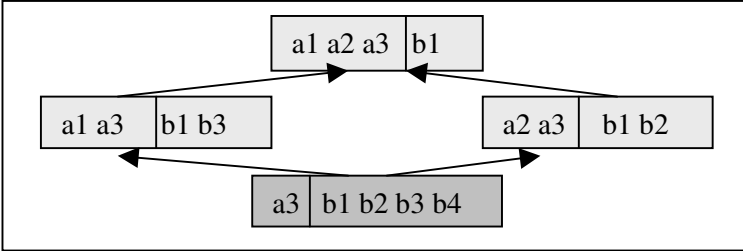


Figure 4.5 : Brute Inheritance Graph G_3

The treatment of the fourth line $L_4 = \{ \{a4\}, \{b1, b2, b3, b5\} \}$, basing on the graph G_3 and with the new node $N' = (\{a4\}, \{b1, b2, b3, b5\})$, permits us to have a value of Int equal to $\{b1, b2, b3\}$. The condition 26 is not satisfied and this value can be deducted from the parents nodes ; it is equal to the union of the ranges of the two father nodes $(\{a1, a3\}, \{b1, b3\})$ and $(\{a2, a3\}, \{b1, b2\})$. Thus no treatment is done and we pass to the following node $(\{a2, a3\}, \{b1, b2\})$ which gives an intersection equal to

its range ; this node causes a modification since the condition 21 is true. Its range is modified and marked and the same thing is done on its fathers by executing the instructions 22, 23 and 24.

The passage to the following node ($\{a1,a3\},\{b1,b3\}$) causes as well a modification since the intersection with $f(x^*)$ is equal to its range. The instruction 22 is executed to modify the domain of the node ; however the instruction 24 does not cause any treatment since the father of the node is already marked in the previous modification. Since all nodes are marked and thus treated, the resulting graph is illustrated in the figure 4.6

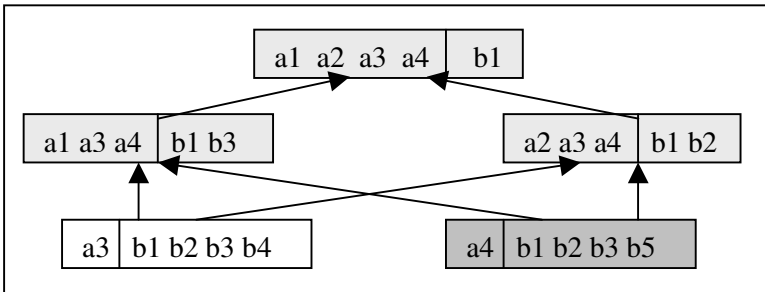


Figure 4.6 : Brute Inheritance Graph G_4

Doing the same treatment for the lines L_5 et L_6 , the graph generated by processing $L_5 = \{ \{a5\}, \{b1, b3, b7\} \}$ is illustrated below :

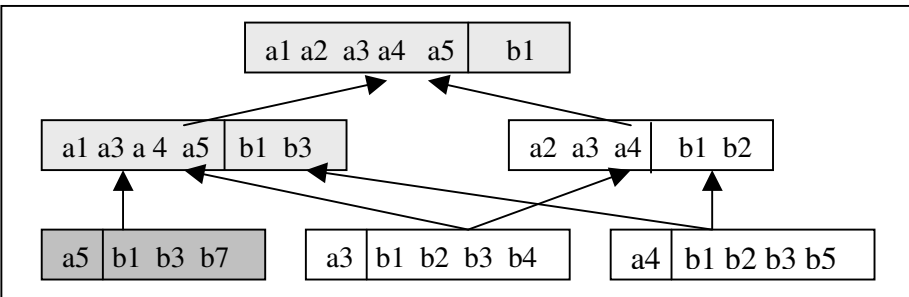


Figure 4.7 : Brute Inheritance Graph G_5

Processing the line $L_6 = \{ \{a6\}, \{b1, b2, b6, b7\} \}$, the binary relation is terminated and the Brute inheritance graph is presented as follows :

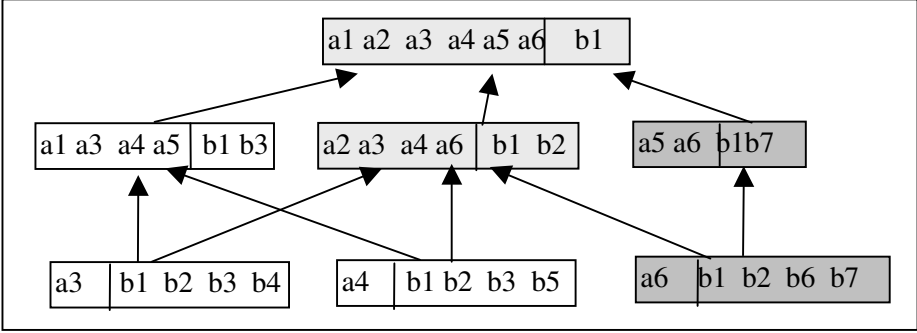


Figure 4.8 : Brute Inheritance Graph of R4

1.2 Simplification of the Brute Inheritance Graph (BIG) : Proper inheritance graph

After the incremental generation of BIG shown in figure 4.8, we remark that there are some necessary simplifications, because for a given optimal rectangle (A,B) to be included in another Optimal Rectangle (A', B'), as defined in section 3.8, it is not necessary to have some redundancy such as elements of A, which are also in A' and elements in B' which are also in B. The simplification consists into the deletion of elements of A from A' and elements of B' from B, because they can be generated dynamically, knowing that the optimal rectangle (A,B) is included in the optimal rectangle (A', B'). This operation is done on all levels of the BIG. As a result, We obtain a simplified brute inheritance graph (see figure 4.10) that is shown to the designer for analyse.

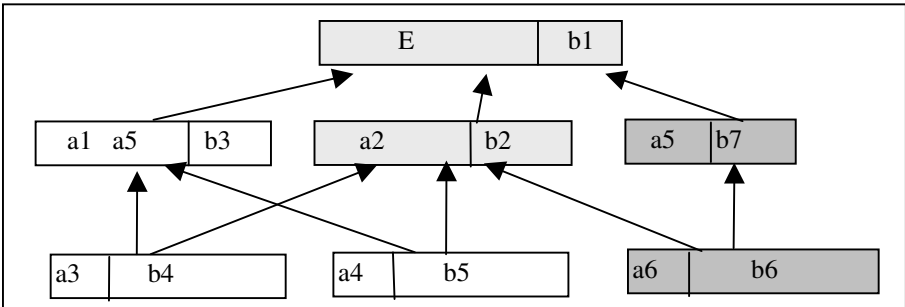


Figure 4.9: Simplified Brute Inheritance Graph

4.1.3. Designer Feedback The difficult task of the designer is now reduced to take decision according to the cases found in the simplified brute inheritance graph. We detect three cases :

1. The cardinality of an optimal rectangle is equal to 1; then we can consider that the name of this entity is the name the class, whose properties are defined by the range of this optimal rectangle. The algorithm Designer-feedback is represented in [16].

2. The cardinality of an optimal rectangle is superior to 1 then we consider two possibilities:

2.1 Elements of an optimal rectangle does not exists in another optimal rectangle ; then we interpret that one of these entities can represent the other and then is indicated as the name of the class. We call these entities Synonymous classes.

2.2 There is an intersection between the optimal rectangle domain and other optimal rectangle domain, then the designer has to verify and resolve the problem of illegitimate inheritance [2], which means that same entities inherit different properties.

3. Other cases could appear, they remain under the responsibility of the designer who is able to decide which modifications are necessary in the graph and does them in a tabular manner in the initial binary matrix. After doing modifications (e.g. add new class, remove a class, or properties) he can interact with the system as explained in the figure 4.1 until he obtains a proper inheritance graph (see figure 4.11).

In our example, the class Object which corresponds to the supremum of the optimal inheritance graph has as a property «Name». This case occurs when all classes, that are linked with Class object (Staff, Student, and Bad_staff) has common properties, they are placed into the range of the supremum.

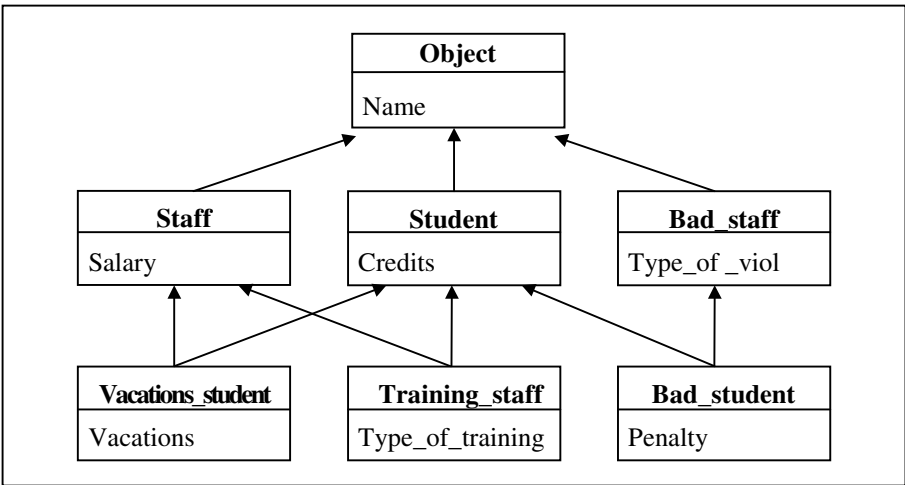


Figure 4.10 : Optimal Inheritance Graph.

4.2. Complexity analysis

We have developed in the application the algorithm of Godin as well as our heuristic. This choice is not neutral. It is based on the fact that the heuristic proposed has the same principle that the algorithm of Godin, it differs by some concepts used.

4.2.1 Algorithm of Godin The complexity of the algorithm of Godin is in the order $O(|G| 2^k)$ whose $|G|$ is the number of the nodes of graph G and k is the limit of $f(x^*)$. [19][21]

Note that the complexity of the creation of the intersection and that the verification of the existence of the intersection in G^* are the factor major of the analysis of the complexity of this algorithm. The complexity of the modification of fishbones between the nodes during of a creation of a new node is neglected [21].

4.2.2 Our heuristic : DRRB_INC As the algorithm of Godin, we are going to calculate the global complexity C_{global} of DRRB_INC for an addition of $(x^*, f(x^*))$ to the graph G. (Remind that R is a binary relationship of E in F).

For an addition of $(x^*, f(x^*))$ to G, we have pass by next phases :

- Phase - 1 : to initial states of the nodes of the graph. This phase has a complexity in the order $c1 = O(\|G\|)$.
- Phase - 2 : After the state initialisation, we verify the existence of $f(x^*)$ in F (range of R). This phase has an equal complexity to $c2 = O(\|f(x^*)\|. \|F\|)$.
- Phase - 3 : Note $c3$ the complexity of this phase that we are going to calculate it. In this phase, we pass to cover the nodes of the graph G.

Each node treaty can undergo one next actions :

- Case - 1 : a modification of the area of the node treaty : this modification will be thereafter propagated on the totality of the nodes fathers of this node. In the bad of cases, a node possesses $(\|G\|-1)$ fathers. Therefore the complexity of the propagation of the modification is equal to $c4 = O(\|G\|-1)$.
- Case - 2 : calculation of the intersection 'Int' of $f(x^*)$ with the range of the node met. In the bad of cases, the range would be equal to F. Where the complexity of this operation evaluated at $c5 = O(\|f(x^*)\|. \|F\|)$.

The node treaty can then be modified as it can provoke a creation of an other node :

- * Case of modification : the complexity is equal to $c6 = O(\|G\|-1)$
- * Case of creation : a node is created after having verified that 'Int' is not included in the meeting of the ranges of fathers of node treaty. This operation has an equal complexity to $c7 = O(\|Int\|. \|F\|)$ since in the bad of cases the meeting of the ranges would be equal to F.

After the creation, the procedure of simplification of the graph is released. Given that the nodes of G are covered to verify if the range of each node is included in the meeting of the ranges of its fathers, the complexity of the procedure of simplification is then equal to $c8 = O(\|G\|. \|F\|^2)$.

The complexity of case - 2 is then :

$$\begin{aligned} c9 &= c5 + \text{Max}(c7, c8) \\ &= O(\|f(x^*)\|. \|F\|) + \text{Max}(O(\|G\|-1), O(\|G\|. \|F\|^2)) \\ &= O(\|f(x^*)\|. \|F\|) + O(\|G\|. \|F\|^2) \end{aligned}$$

The complexity of the phase - 3 is therefore :

$$\begin{aligned} c3 &= \|G\| + \text{Max}(c4, c9) \\ &= \|G\| + \text{Max}(O(\|G\|-1), [O(\|f(x^*)\|. \|F\|) + O(\|G\|. \|F\|^2)]) \\ &= \|G\|. [O(\|f(x^*)\|. \|F\|) + O(\|G\|. \|F\|^2)] \\ &= O(\|G\|. \|f(x^*)\|. \|F\|) + O(\|G\|^2. \|F\|^2) \end{aligned}$$

We notice that the complexity of the simplification is the factor major of the total complexity.

The complexity of the heuristic DRRB_INC is then :

$$\begin{aligned} C_{\text{global}} &= c1 + c2 + c3 \\ &= O(\|G\|) + O(\|f(x^*)\|. \|F\|) + O(\|G\|. \|f(x^*)\|. \|F\|) + O(\|G\|^2. \|F\|^2) \\ &\sim O(\|G\|^2) \end{aligned}$$

Since our heuristic gives a number of node smaller than that given by the algorithm of Godin (that is proven experimentally), then the complexity of our heuristic is lesser than the algorithm of Godin : $O(\|G\|^2) < O(\|G\| \cdot 2^k)$. This result appears unexpected. Indeed, our heuristic proposed suggests the simplification of the graph, that is the factor major in the calculation of the complexity, during each creation of a new node. This simplification is left in the algorithm of Godin. But we do not forget that the cardinality of the graph is the main term in the two complexities. *Remark* : The software environment adopted for the realization of the application of INC-RDBR is the Visual Basic version 5 under Windows 95. We remark that more the relationship is intense, more our heuristic is best in time of execution. It imports to signal that our heuristic manages an optimum number of comparative rectangles to the number of initially given entities.

4.2. Comparison study

For the same example of the binary relation (figure 4.2), the iO2 method in [30], based on the structure the of the Galois Lattice, gives rise to nine (9) maximal rectangles [16]. This method is not optimal because the number of maximal rectangles corresponding to classes are superior to the initial number of entities defined in the real world (6). In addition the iO2 method is not incremental : adding an entity (a line to the binary relation) causes retrieval of maximal rectangles by considering the new version of the relation (with 7 lines and not with six). With the method of RDBR in [16], only six optimal rectangles are generated. This number is equal to the number of entities defined in the real world. RDBR is optimal, however, it is not incremental as in the case of iO2. The proposed algorithm (INC_RDBR) gives the same number of rectangles as DRBR and it is incremental.

Inheritance graph Generation Method	Binary Relation of 16 pairs (6 entities)	Advantages (+)
	Number of generated nodes	Inconvenient (-)
iO2	9 nodes	- Non incremental
RDBR	6 nodes	- Non incremental
INC-RDBR	6 nodes	+ Incremental

It is not currently that the number of Optimal rectangles generated by RDBR is less than the number of maximal rectangles which constitutes the nodes of a Galois Lattice generated by [29] corresponding for the same example (see figure 4.11 and figure 4.12).

The results obtained by [29] for the example illustrated by figure 4.3 shows nine nodes of Galois Lattice. With RDBR, we avoid the redundancy represented by three rectangles drawn in bold in figure 4.11. Furthermore, we ensure that we do not lose information because INC-RDBR computes the cover of R. Another advantage of our

method is that we achieve an important gain in the number of concept when the cardinality of the binary relation is high. The proof that we obtain all times better results (in relation to the number of nodes) than methods used for generating Galois Lattice is expressed by the fact, that from a set of maximal rectangles which contains the same couple (a, b) , we extract only the maximal rectangle which ensures maximal gain as is shown in section 3.3. In the worst case, we have for a given couple (a, b) of R only one maximal rectangle. Then the minimal cover of R is a set of maximal rectangles which could be organized into a Galois Lattice.

5. Conclusion

We introduced a formal method for a semi automatic and incremental generation of a proper inheritance graph which represents the class hierarchy of an object oriented conceptual scheme. Our method has a mathematical foundation and allows us to obtain a number of classes, less than the number of maximal rectangles given by methods which use Galois Lattice structure as a support. In addition, it gives the same results as RDBR.

A new polynomial algorithm is introduced to generate, organize and simplify incrementally optimal rectangles to construct a proper inheritance graph hierarchy. The feedback of the designer which is crucial in the analysis and design steps is taken into account. INC-RDBR can be used in several domains such as Documentary Database and in Artificial Intelligence for automatic consensual rules generation from multiple experts.

8. Bibliography

- [1] Ann L. Winblad, Samuel d. Edwards, David R. King - «Object-Oriented Software»: Addison-Wesley, 1990.
- [2] M. Armstrong, Richard J. Mitchell, «Uses and Abuses of Inheritance» Journal Engineering Software, Vol 9 (1) 19 - 26, Jan 1994,
- [3] M. Barbut, B. Monjardet, «Ordre et classification», Algèbre et combinatoire, Tome II, 1970, Hachette.
- [3] H. Belaid, Ali Jaoua, «Abstraction of Objects by Conceptual Clustering», In proceeding of Joint Conference on Information Sciences, Durham, USA, 1997.
- [4] N. Belkhiter, C. Bourhfir, M.M. Gammoudi, A. Jaoua, N. Le Thanh, M. Reguig, «Architecture rectangulaire optimale d'une base documentaire indexée», Université Laval, Département d'Informatique (Rapport de Recherche), DIUL-RR-9201, 1992.
- [5] N. Belkhiter, C. Bourhfir, M.M. Gammoudi, A. Jaoua, N. Le Thanh, M. Reguig «Décomposition Rectangulaire Optimale d'une Relation Binaire: Application aux Bases de Données Documentaires». Information Systems and Operational Research Journal, 32 (1): 33-54, 1994.
- [6] N. Belkhiter, J. Desharnais, G. Ennis, A. Jaoua, H. Ounalli, M.M. Gammoudi, «Formal Properties of Rectangular Relations». In Proceeding of the Ninth International Symposium on Computer and Information Science, ISCIS IX, Antalya, Turkey, November 7-9, 1994.
- [7] G. Booch, «Object-Oriented Analysis and Design with Applications», Second Edition - The Benjamin/Cummings Publishing Company, Inc.1994.

- [8] J.P. Bordat, «Sur l'algorithmique combinatoire d'ordre finis», Thèse de Doctorat d'état de l'Université de Montpellier II, (USTL), Avril 1992.
- [9] J. Daniels, «Objects Design Methods and Tools», IEEE London, UK, 1994.
- [10] D. An Chiang, Ming-Chi Lee, «Cyclic Inheritance Detection for Object-Oriented Database», IEEE New York, NY, USA, Vol 2 1047 pp., 1992, Univ. TAIWAM.
- [11] Chi-Ming Chung; Lee M. C, «Inheritance-Based Object-Oriented Software Metrics», IEEE New York, NY, USA, 1992 Vol 2 1047 pp.
- [12] Chi-Ming Chung ; Chun-Chi Wang, Lee M. C, «Class Hierarchy Based Metric for Object-Oriented Design», IEEE New York, NY, USA, 1994 Vol 2 xxvii+1111 pp.
- [13] M. M Gammoudi. «Méthode de décomposition rectangulaire d'une relation binaire: une base formelle et uniforme pour le génération automatique des thesaurus et la recherche documentaire. Thèse de Doctorat, Université de Nice Sofia Antipolis, Septembre 1993.
- [14] M.M. Gammoudi. S. Labidi «An Automatic Generation of Consensual Rules Between Experts Using Rectangular Decomposition of a Binary relation». In Proceeding of the XI Brazilian Symposium on Artificial Intelligence, Fortaleza, October 1994.
- [15] M.M. Gammoudi «Heuristics for Clustering Method and its use in Information Retrieval System», in Proceeding of the International Conference SCCC'96, Valdivia, Chile, November 1996.
- [16] M. M. Gammoudi, J. D. Mendes, W. S. Pinto "Inheritance Graph Hierarchy Construction using Rectangular Decomposition of a Binary Relation and designer feedback" in Proceedings of the International Conference on Conceptual Modeling ER'97. Los Angeles – November 1997.
- [17] B. Ganter, "Two Basic Algorithms in Concept Analysis", Preprint 831, Technische Hochschule Darmstad, p. 28, 1984.
- [18] M. R. Garay, D. S. Jonhson, «Computer and Interactability : A guide of the theory of NP-Completeness.», W. H. Freeman, 1979.
- [19] R. Godin, R. Missaoui, H. Alaoui, «Learning Algorithms Using a Galois Lattice Structure», in Proceedings of the 1991 IEEE Int. Conf. On Tools for AI, San Jose CA-Nov, 1991
- [20] R. Godin and H. Mili, «Building and Maintaining analysis-level class hierarchies using Galois Lattice», OOPSLA-93, Washington DC, USA, ACM press pp. 394 – 410, October 1993.
- [21] R. Godin and H. Mili, «Conceptual Clustering Methods based on Galois lattice and applications», French Review on Artificial Intelligence, Vol 9 N 2, 1995 pp. 105-137.
- [22] A. Guenoche, «Construction du treillis de Galois d'une relation binaire», *Math. Inf. Sci. hum.*, 28ème année, (109) : 41-53, 1990.
- [23] J. Mayrand, G. François, M. Ettore. «Inheritance Graph Assessment Using Metrics», Proceedings of METRICS '96, 1996 IEEE.
- [24] E.M. Norris, «An algorithm for Computing the Maximal Rectangles in a Binary Relation», *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2) p. 243-250, 1978.

- [25] R. E. Stepp and R. S. Michalski, «Conceptual clustering: inventing goal-oriented classifications of structured objects», R. Michalski, J. Carbonnel, and T. Mitchell, (eds)., *Machine Learning: An I.A. Approach*. San Mateo, Calif. Morgan Kauffmann, 1986.
- [26] I. Schmitt, G. Saake, «Integration of inheritance trees as part of view generation for database federations», In proceeding of the 15th International Conference on Conceptual Modelling - ER'96. Cottbus, Germany, 7th - 10th October 1996.
- [27] W. Sun, S. Guo, F. Araf, S. Tu «Supporting Inheritance in Relational Database», *IEEE Computing* 1992.
- [28] R. Wille, «Finite Distributive Lattices as Concept Lattices», *Atti Inc. Logica Matematica*, (2) 1985.
- [29] R. Wille, «Knowledge Acquisition by Methods of Formal Concept Analysis», In E. Diday (Eds). *Data Analysis, Learning Symbolic and Numeric Knowledge*, NY, 1989.
- [30] A. Yahia, L. Lakhal, JP. Bordat, «Designing Class Hierarchies of Object Database Schema», in *Proceedings of the Computer journal*, 1997.

Developing an Object-Oriented Video Database System with Spatio-Temporal Reasoning Capabilities

Shermann Sze-Man Chan and Qing Li

Department of Computer Science, City University of Hong Kong

Tat Chee Avenue, Kowloon, Hong Kong, China

{shermann, csqli}@cs.cityu.edu.hk

Abstract. Video data management (VDM) is fast becoming one of seminar topics in multimedia information systems. In VDM systems, there exists an important need for novel techniques and mechanisms which can provide an efficient retrieval facility of the voluminous data stored in the databases. While "content-based" retrieval provides a direct and an intuitive approach for video data access, it is inadequate from efficient video data management viewpoint. This is because many (semantic) features of video data cannot be extracted out from the video itself automatically; moreover, video objects may share annotations or descriptions. To address these problems, we have been developing an object-oriented video database system which can complement content-based access with high level (declarative) query-based retrieval. As video objects by their nature are rich in spatial and temporal semantics, a versatile modeling mechanism is devised to support Spatio-Temporal reasoning. Associated language facilities are also being developed to accommodate a wide range of video queries and annotation activities.

1 Introduction

Nowadays, due to the rapid progress in video technology, large amounts of video sources have become available. Video data management is fast becoming one of the seminal topics in multimedia information systems. An important issue to be addressed in a video database management system is video retrieval. Content-based retrieval [IEEE1995], which uses image features such as colour, texture and shapes, is alone not sufficient to fulfil the user's need. The main reason is due to the fact that many high-level (semantic) features of video data are extremely difficult (if not impossible) to be extracted automatically from the video itself; moreover, video objects may share annotations and/or descriptions. Consequently, it is necessary and cost-effective to complement content-based retrieval with declarative, efficient query-based retrieval in the video database system. As the key characteristics of video data are the spatial and temporal semantics associated with it, this makes video retrieval quite different from, and more challenging than, other types of data access such as text, voice or image retrieval.

In this paper, we describe a video database management system which supports video data Spatio-Temporal reasoning through a versatile modeling mechanism called CCM/ST, along with an associated query language (CAROL/ST) and a specification language. In particular, CCM/ST is based on its previous research project (CCM) that we have developed earlier [LH1995, LS1992]. CCM offers a number of versatile features such as dynamic grouping of scenes and multifaceted features modeling; these features have shown to be extremely useful for video manipulations [LL1995, LLX1997]. The semantic meanings captured by the scenes and features also facilitate declarative retrieval, which results in a high-level query language called CAROL [LYC1998] which was developed to provide flexible video-retrieval in both context-dependent and context-free manners [LL1996]. However, neither CCM nor CAROL has (explicit) spatial and temporal facilities for modeling and accessing video data, hence further extensions are needed to our model and language in order to support more effective and efficient video retrieval.

At this stage, our main emphasis is on the feasibility of developing the new features within CCM, both in terms of expressiveness and implementability. In next section, we review our existing prototype system and its underlying model (i.e. CCM) with respect to the scene and feature facilities. In section 3, we describe the new modeling facilities, along with a brief look at its associated query language CAROL/ST. We describe in section 4 some implementation issues on the index mechanism of supporting spatio-temporal reasoning; moreover, an Activity modeling technique based on the spatio-temporal reasoning is also presented. Finally, conclusions and future work are given in section 5.

2 Background

A current trend to database management is towards object-oriented database (OODB) management systems, where each real world entity is modeled as an object and each of them has a set of attributes and methods. The most important features of an OODB model include the inheritance mechanism as well as complex/composite object modeling [KBG1989]. As video data is considerably more complex than other types of data such as text, voice and images, it is much more suitable to use OODB systems to model and manage these data types.

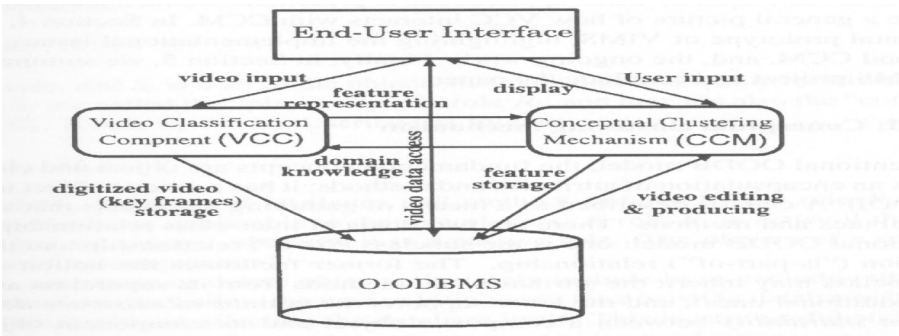


Fig. 1. A general Video DBMS architecture

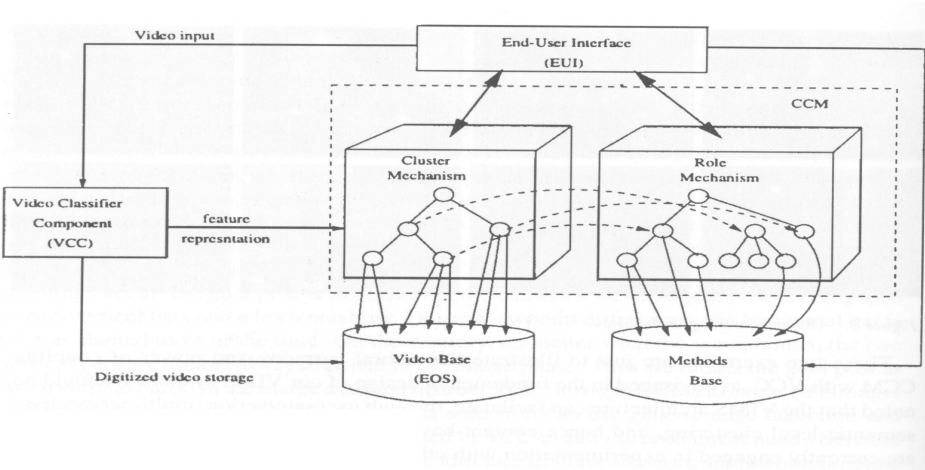


Fig. 2. CCM internal representation of Scenes and Features

Fig. 1 shows the general architecture of an object-oriented video database management system that we have developed [LLX1997] to support the accessibility and manipulability of video data. The Video DBMS employs two fundamental components: a Video Classification Component (VCC) for the generation of effective indices necessary for structuring the video data [LLX1997], and a Conceptual Clustering Mechanism (CCM) having extended object-oriented features and techniques [LL1995, HLLX1996]. Fig. 2 illustrates the internal representation of CCM by using Cluster (Scene order) mechanism and the Role (Feature index) mechanism in a more detailed manner. By incorporating CCM concepts and techniques together with the classified features and indices generated from the VCC, our system enables users to form dynamically video programs from existing video objects based on semantic features/index terms.

In order to retrieve video data stored in the video database, a declarative video query language (CAROL) has been designed and implemented [LYC1998]. Though both CCM and CAROL offer a rich set of capabilities that are seldom available from other video database models and systems, there are some deficiencies with respect to the spatio-temporal reasoning. Therefore extensions are needed to both the model as well as the language.

3 Our Approach

In order to make our Video DBMS to be more effective and of more features and functions, its architecture is extended and enhanced as shown in Fig. 3. Specifically, both the CCM and the Query Language Processing components are now enhanced to support spatial and temporal semantics of video data, with CCM becoming CCM/ST, and CAROL becoming CAROL/ST as a consequence. Beside these two components, we also add a *specification language processing* component into it for expert-users to annotate some conceptual features which include time- and space-dependent information such as event, action, and movement of objects, etc. as additional

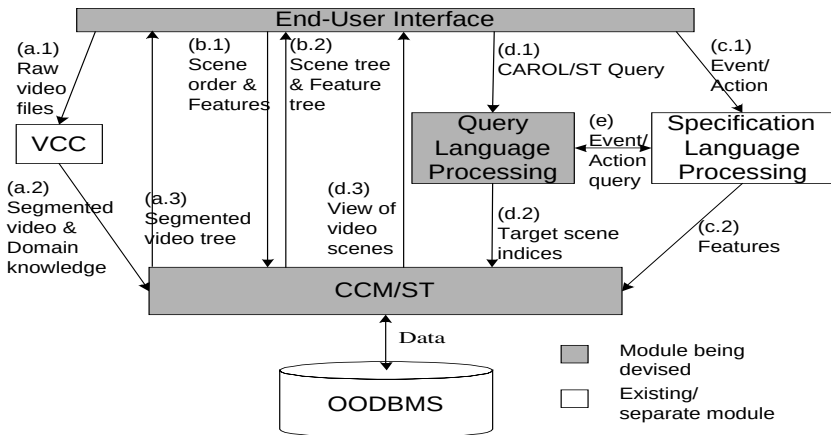


Fig. 3. Relationships among components in the Video DBMS

indices of video data to be used by CAROL/ST. This specification language will also serve as an interface to (future) computer vision and image processing algorithms that may extract such information items automatically.

Fig. 3 shows (a) the process of video classification and segmentation from (a.1) to (a.3); (b) the process of adding features into CCM/ST, both general and spatio-temporal, and building Scene-order/Feature-index trees from (b.1) to (b.2); (c) the process of adding spatio-temporal features into CCM/ST by an Event/Action model from (c.1) to (c.2); (d) the process of querying

from (d.1) to (d.3). If the user makes an Event/Action query, the process of query is from (d.1), and then to-and-from (e), then to (d.2) and (d.3).

3.1 CCM/ST: an extended object-oriented mechanism supporting spatio-temporal semantics

3.1.1 Supporting Scene Ordering

In the original CCM model [LL1995, HLLX1996], cluster is a conceptual video scene collection and it consists of attributes, methods, and dynamic groupings of video objects in database, within which each group of video objects is linked with one role. Since meaningful video sequences are identified and segmented by VCC [LLX1997], conceptual video objects are created and associated with their describing data incrementally and dynamically. Similar to cluster, role is a feature/index of video scene and it consists of attributes and methods. Clusters only include-in or exclude-out objects of database and allow user to form views over existing database. Therefore a cluster can be used to represent a Scene of a video program, and a role can be employed to represent a Feature/index within the scene. Specifically, a Scene *S* is a dynamic object that has a tripartite form:

$S = \langle A, M, X \rangle$

where *A* is a set of scene attributes, *M* is a set of scene methods, and *X* is a set of the Feature-Video associations:

$X = \{ \langle F_i, V_i \rangle \mid 1 \leq i \leq n \}$

where *F_i* is a feature, and *V_i* is a set of video objects that carry the feature within the scene. Hence, the objects in *V_i* are also called the “video players” of the feature *F_i*, and they are also the constituents of the scene *S*. A feature *F_i* can be described as follows:

$F_i = \langle A', M' \rangle$

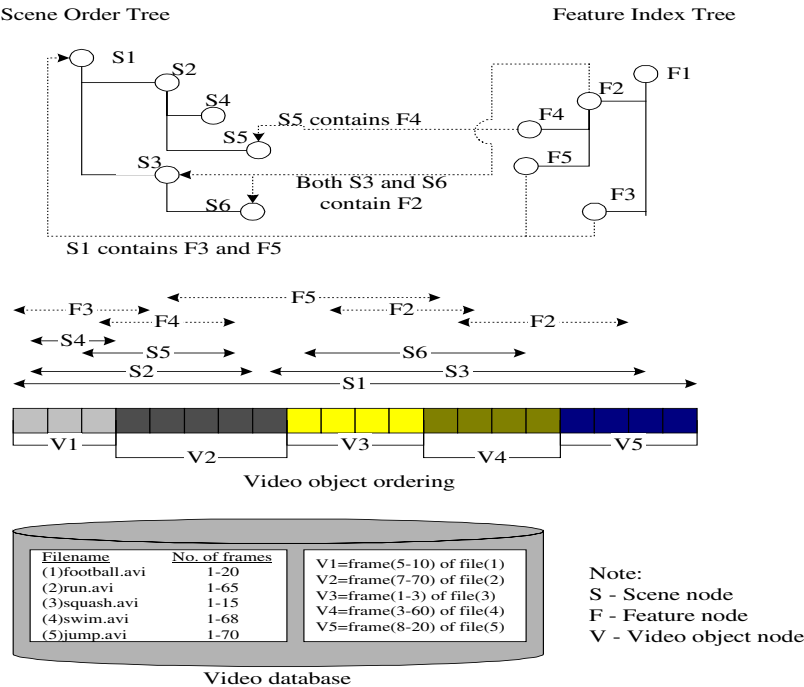


Fig. 4. A video program described by the scene order and feature index trees in CCM/ST

where A' and M' are a set of attributes and methods defined by the feature F_i ; these attributes and methods are applicable to the video players of the feature. In our Video DBMS, a feature is the same as a semantic descriptor (a glossary term) that is of interest to end-users.

For video applications, an important concept to support is Table of Contents (ToC) which describes the structural as well as sequencing relationships of video data within a video program. An example is intuitively illustrated in Fig. 4. There are two kinds of relationships embedded in the scene order tree: inheritance and composition, but only inheritance is embedded in the feature index tree. Sub-scene can choose to inherit attributes and methods from its super-scene and it is similar to the case of sub-feature and super-feature. However, in order to support ToC, super-scene needs to have composite structure to contain sub-scenes. The scene order tree in Fig. 4 has six nodes. S_1 is the root node and it contains a set of scenes, and forms a ToC. It has two immediate sub-scenes, S_2 and S_3 . For those sub-scenes are at the same level, they can be overlapped such as S_4 and S_5 . All the sub-scenes would be the subset of their immediate super-scenes. Each scene node may or may not contain a feature, and a feature node may or may not be linked by any scene nodes. Meanwhile, S_1 has features F_3 and F_5 , S_5 has F_4 , and both S_3 and S_6 share the same feature F_2 . Within a Scene-Feature-Video association, the video objects linked by the feature should form a subset of the *constituents* included by the associated scene. In order to support ToC in CCM, the notion of scene is extended by defining a temporal scene S_i as follows:

$$S_i = \langle A, M, X, T \rangle$$

where A and M denote attributes and methods defined by the scene S_i respectively, X is a set of the Feature-Video associations, and T is a template of showing the temporal ordering of the video objects in the scene S_i . Within the set of Feature-Video associations, all the video objects of each pair of the associations should form a subset of the constituents of a scene S_i , so as to avoid that a scene contains some Feature-Video associations which do not relate to the scene. Also, in order to maintain the ToC, all the constituents of the sub-scenes should form a subset of their immediate super-scenes.

Note that ToC should include all the video objects of the Feature-Video associations within the Scenes. Currently, only one temporal operator [CL1999] is considered so as to obtain a sequential temporal ordering of video objects within a scene order tree. To provide a flexible environment to the users in future, it is possible to include more than one identical video objects in ToC and to apply different types of temporal operators (ref. [CL1999] for details)^Ψ.

3.1.2 Supporting Spatial and Temporal operations

In CCM/ST, a list of operators and functions concerning spatial and temporal semantics [CL1999] is chosen to support. There are five basic spatial functions that are applicable for comparing two features, which contain some spatial information. In addition, there are six 1-D and nine 2-D spatial operators that are used to identify the position of an image object in a low-level format. The objects together with their positions are stored in the database in a specific data structure [CL1999].

Beside the issue of spatial operations, there are several types of temporal operators and functions [SHC1998], e.g. interval specification functions, interval ordering functions, interval ordering operators, and interval comparison operators in CCM/ST [CL1999]. Two kinds of time dimensions i.e. valid-time and user-defined time, are applicable to both "Object level" and "Attribute level" [TCG1993]. Therefore, CCM/ST can store the past histories and future plans of scenes and features, and thus can provide different versions of a video program.

^Ψ Therefore, it is not restricted to sequential operator only. In fact, several videos can be shown at the same time.

3.2 CAROL/ST: a query language supporting spatial and temporal primitives

As mentioned earlier, our Video DBMS has a query language (CAROL) [LYC1998] which is devised based on CCM features. The structure of CAROL is similar to that of the standard Structured Query Language (SQL) and it supports five primitive video retrieval functions, namely:

- *select scenes/features/videos by their attribute restrictions*
- *select video players (video objects) by scene's restriction*
- *select scenes by their features' restrictions*
- *select features by scenes' restrictions*
- *select scenes by their video players' restrictions*

Since only three types of objects (scene, feature, and video player) are used in the database, declarative SQL with some extensions [LYC1998] is sufficient to make queries in our system. As supporting spatial and temporal semantics, CAROL/ST should be extended to provide a more expressive and flexible query language. In particular, the introduced temporal and spatial operators of CCM/ST [CL1999] can be incorporated into CAROL/ST naturally, as the following query examples demonstrate.

(A) Selection concerning Valid-time:

1. Consider that a user wants to retrieve all the video objects about "SportsNews" before a given date (say, current date). The query can be specified by the following CAROL/ST statement (x is a set of scene objects)^φ:

```
SELECT x FROM SCENE y
WHERE y HAS FEATURE SportsNews
AND VT_INTERVAL(y) BEFORE CURRENT_DATE;
```

All video objects are extracted from the result of the above query, x, which contains the feature "SportsNews" and the valid-time of each of the retrieved scenes is before current date. Therefore, "histories" of scenes about SportsNews can be retrieved.

2. Consider another user wants to get the earliest made and indexed (feature) piece(s) of video objects from a scene "MTVCollections". The query can be represented by the following query (x is a set of feature objects)^λ:

```
SELECT x FROM FEATURE y
WHERE y HAS SCENE MTVCollection AND y = FIRST(y);
```

The above query can be divided into three parts. First, it is to retrieve all features contained in a scene of "MTVCollection". Then, each of the retrieved features is next sorted by its valid-time (in an ascending order by default). Finally, the query result is those video objects of the features that rank the first among the sorted ones.

(B) Selection concerning Spatial/Temporal semantics:

1. The following query is to retrieve a feature named "Football" from a scene EveningNews, and the feature is located at the top of the left-hand side of some video clips for at least 60 frames (i.e., about 2 seconds) (x is a set of feature objects).

```
SELECT x FROM FEATURE y
WHERE y HAS SCENE EveningNews
AND y.name = "Football" AND y AT TOP_LEFT FOR >= 60;
```

2. To specify a query which is to retrieve some video clips showing that Clinton is with Lewinsky for no less than 4 seconds, the following CAROL/ST statement can be formulated (x is a set of scene objects):

^φ x is selected from scene objects, therefore it is a set of scene objects.

^λ x is selected from feature objects, therefore it is a set of feature objects.

```

SELECT x FROM SCENE y
WHERE y HAS FEATURE Clinton AND y HAS FEATURE Lewinsky
AND F_INTERVAL(Clinton) EQUAL F_INTERVAL(Lewinsky)
AND DURATION(F_INTERVAL(Clinton)) > 120;

```

The query above retrieves a scene, which has features “Clinton” and “Lewinsky”, and the two features have the same interval of the same video object with more than 120 frames (about 4 seconds). Note that the above query is an example of *imprecise* query, in that it may actually match more than one type of events (including, e.g., Clinton walking with, talking with, eating with, Lewinsky, etc.).

3. The query below is to retrieve a scene that has features “Sun” and “Sea”, and the two features exist together at a certain number of frames where “Sun” is at a position on top of “Sea” (x is a set of scene objects).

```

SELECT x FROM SCENE y
WHERE y HAS FEATURE Sun AND y HAS FEATURE Sea
AND F_INTERVAL(Sun) OVERLAP F_INTERVAL(Sea)
AND Sun = TOP(Sun, Sea);

```

3.3 Specification Language

As shown in Fig. 3, a specification language (or content/semantics description definition language) is to be used by expert users to annotate user-interested Event/Action semantics into CCM/ST. It is based on the spatio-temporal reasoning, which is embedded in the Feature Index Hierarchy. In our Video DBMS, indices are generated by VCC, and are grouped and linked by the CCM/ST. The indices are normally the key frames of the video objects. In order to extract and compare the image features of the video objects spatially, the existing index mechanism needs to be enhanced.

As the current technology on feature extraction from images is still limited and developing, a new data structure, ST-Feature [CL1999], is thus introduced to CCM/ST to accommodate possible space-/time- dependent annotations. The structure of the ST-Feature is as follow:

ST-Feature = <{Position-array, Start-frame, End-frame}>

where Position-array is the spatial representation of the image feature, and together with Start-frame and End-frame store the duration of the feature denoted by ST-Feature. The implementation detail of ST-Feature is described in [CL1999]. Note that ST-Feature is used when an image object can be identified from the video objects. Besides static object, ST-Feature can represent a moving object too. Thus, if there are some features in the same scene, by using an Activity model shown in Fig. 5, we can conclude that some events and actions happened in the video object.

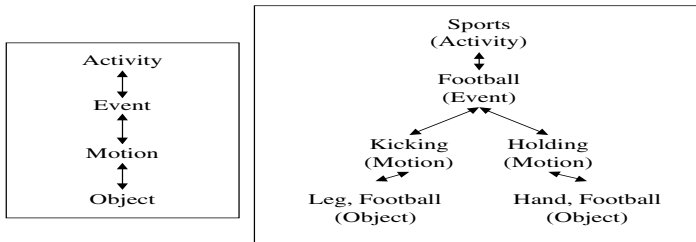


Fig. 5. Activity Model and a “Football” example

In Fig. 5, the left-hand side is the architecture of the Activity Model and right-hand side is a “Football” example. The model consists of four levels, Activity, Event, Motion, and Object. The top three levels are mainly used by the query language processor to reason what activity the

user wants to retrieve, so that the processor would retrieve the video scene from the database according to the objects (i.e., features with spatio-temporal semantics) from the bottom level of the model.

A user may input an activity into the Specification Language Processing Component (shown in Fig. 3(c.1)) by using terms such as “playing sports”, “playing football”, “kicking football”, or “kicking”. More specific terms would yield more specific retrieval results. In general, the first word is known as a verb and the word followed is a noun. The processor would analyze from the Motion level first. After some keywords are matched, the processor would search up to the Event level using the second word. For example, if the term is “kicking football”, the processor searches “kicking” from the Motion level, and then uses “football” to search from the Event level. If the term is “playing football” and there is no “playing” in the Motion level, the processor will try to reason the thesaurus of the word and then search again. However, if there is no match of word from the model, the processor would just skip the “verb” and search the “noun” from the Event level to the Activity level. After the threshold of the search is met, the processor would go down to the corresponding Object level. Then it would input those objects from the Object level into the Feature Index Tree as features and ask the user to input some spatio-temporal semantics (ST-Feature) into the database (shown in Fig. 3(c.2)).

At a later time, user may want to retrieve video data based on some activities from the database. For example, he may input an *activity query* like “kicking football”. The Query Language Processor first gets some collections of objects from the Activity Model (shown in Fig. 3(e)) and then retrieves the result as the original query processing (CAROL/ST) by treating the collections of objects as Features and ST-Features. Therefore, the main concern of the Activity model is on how to include all common and significant activities.

4 Implementation Issues

4.1 Query Indexing Mechanism

With the spatio-temporal semantics added into the system, users may now impose some common queries such as asking for the spatial locations, the size of a feature, and comparing two or more features in terms of position and direction. In particular, the following types of user queries are possible:

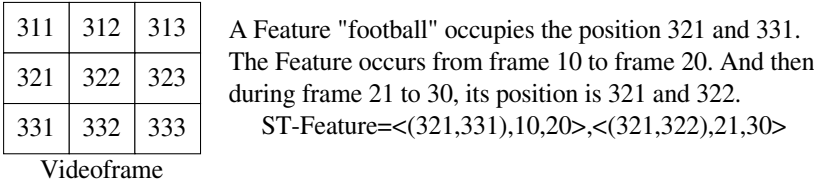


Fig. 6. The data structure of ST-Feature

- (A) Specify the spatial location
 - 1. “What feature is at left?”
 - 2. “What feature is at top and left?”
 - 3. “Is the feature at right?”
 - 4. “Is the feature at bottom and right?”
 - 5. “What is the position of the feature?”
- (B) Specify the size of the feature
 - 1. “Does the feature have the size of three cells?”
 - 2. “Does the feature have the size of more than three cells?”
 - 3. “What features have the size of four cells?”
 - 4. “What features have two or less number of cells?”

(C) Compare two or more features

1. “Which feature is on top of the other, ST1 or ST2?”
2. “What feature is on top of ST3?”
3. “For the feature ST1, what else features are at the same video frame?”

In order to facilitate the processing of these queries, we introduce a query indexing mechanism. An example is shown in Fig. 7. Two indexing structures based on the example of Fig. 7 are shown in Fig. 8 and Fig. 9. And finally, Fig. 8 is extended to Fig. 10 to carry out comparison among features in a more detailed manner.

In Fig. 7, there are five continuous video frame intervals. Due to that the features and their positions are the same inside the frame interval, they are grouped together and shown by a single video frame. Each frame is divided into nine cells and the details are described in [CL1999]. Since each ST-Feature may contain a set of “Position(s) and Frame numbers”, “(1)” means the first element of the set, “(2)” means the second element, etc. In Video frame interval 1, there are two Features with Spatio-temporal semantics i.e. ST2(1) and ST3(1), and they occupy the same video frame interval. The position of ST2(1) is represented as 322 and ST3(1) is represented as 331 (cf. Fig. 6).

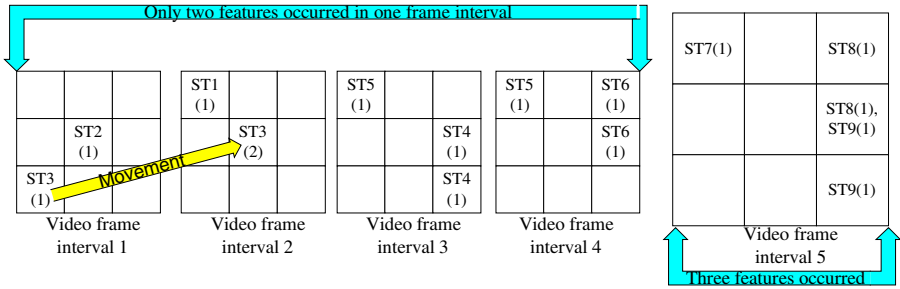


Fig. 7. Spatial relationships among features representing in five continuous Video frames

In Video frame interval 2, there are ST1(1) and ST3(2). ST1(1) is represented as 311 and ST3(2) is 322. ST3 occurs in different positions from Video frame interval 1 to Video frame interval 2, which indicates the Feature has a movement. In Video frame interval 3, ST4(1) occupies two cells i.e. 323 and 333, which indicates that ST4(1) is larger than ST5(1) (the latter is 311). In Video frame interval 4, ST5(1) still exists but its spatial relationship with the feature is changing from ST4(1) to ST6(1). ST6(1) is represented as 313 and 323. And finally, in Video frame interval 5, it shows the relationships among two or more Features, where ST7(1) is 311, ST8(1) is 313 and 323, and ST9(1) is 323 and 333. The former four video frame intervals show the binary relationships of the features and the last one shows a ternary relationship.

According to the spatial representations of the features shown in Fig. 7, the features can be classified and sorted into six linked-lists i.e. HL, HM, HR, VT, VM, and VB. HL means the position of horizontal left; HM means horizontal middle; HR means horizontal right; VT means vertical top; VM means vertical middle; and VB means vertical bottom. For HL, HM, and HR, the lists are used to classify the horizontal position of the features. For VT, VM, and VB, they are used to classify the vertical position of the features. In Fig. 8, since each linked-list represents its specific position, only IDs of ST-Feature (i.e. object IDs) are needed to store in the structure. Therefore, each ST-Feature is stored once in the database and the indexing mechanism only keeps the record of object IDs. Thus disk space required is not much. As a result, the query no.1 and no.2 of section 4.1(A) can be evaluated, whereas the 1-D and 2-D spatial operators [CL1999] can be used to retrieve those features.

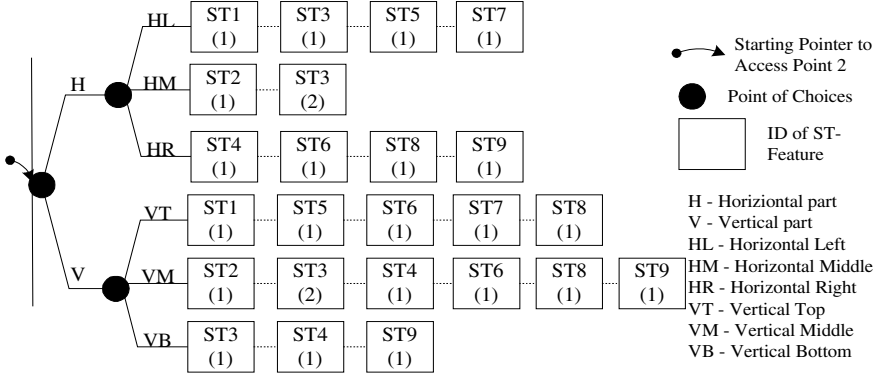


Fig. 8. An example of Access Point Two of the Query Indexing Mechanism

However, in order to evaluate those queries such as query no. 3, 4, 5 of section 4.1(A), the IDs of ST-Feature should be indexed. Since ST-Features are originally sorted and stored in the database as a linked-list, it is only needed to link the structure of Fig. 8 with the original linked-list. Thus the structure of the original linked-list is as shown in Fig. 9. In Fig. 9, beside the ID of ST-Feature and the pointer of Access Point two, each node stores its structure of ST-Feature i.e. $\langle \{Position\text{-}array, Start\text{-}frame, End\text{-}frame\} \rangle$, where the detail is described in [CL1999].

Consider a query to retrieve the scenes that contain a feature “Football”, and the feature contains the spatial semantics where it is at left. The query can be written in CCM/ST as follows (x is a set of scene objects):

SELECT x FROM SCENE y
WHERE y HAS FEATURE Football AND Football AT LEFT;

For query processing, there are two ways to retrieve the result. The first is to search the Feature Football from the Feature Tree. When the Feature is found, check whether the Feature-Video association links it; and check whether it contains the spatio-temporal semantics, ST-Feature. After obtaining the ID of the ST-Feature, enter the Access Point One of the indexing mechanism, and get the pointers of Access Point Two. If there is a HL pointer, it means the Feature is at the left. The second way is to search from the Access Point Two of the indexing mechanism first, and get those IDs of ST-Feature from the HL list. Then to select the Feature Football from the Feature Tree and check whether the Feature contains any spatio-temporal semantics which has the same ID as listed in the HL list.

At this moment, we are not so much concerned with the size of the feature. But the query can specify the size of the feature indirectly as it is supported in the indexing mechanism. (Note that the decision of applying which retrieval way in our above example may depend on the size of the first retrieved result.) The linked-list of ST-Feature shown in Fig. 9 is stored together with the Position array, queries such as section 4.1(B) can also be made as the following example (x is a set of scene objects):

SELECT x FROM SCENE y
WHERE y HAS FEATURE Sun
AND Sun AT TOP_LEFT AND Sun AT TOP_MIDDLE;

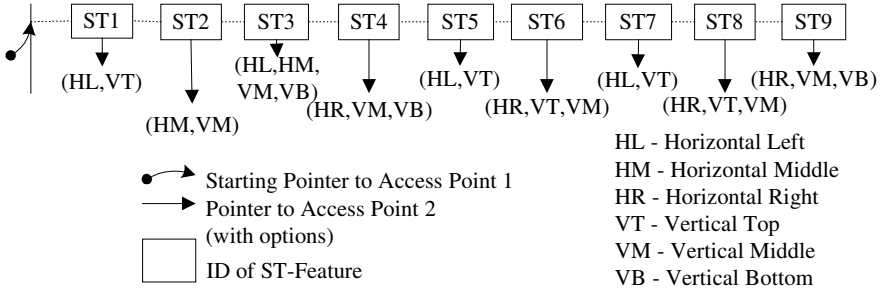


Fig. 9. An example of Access Point One of the Query Indexing Mechanism

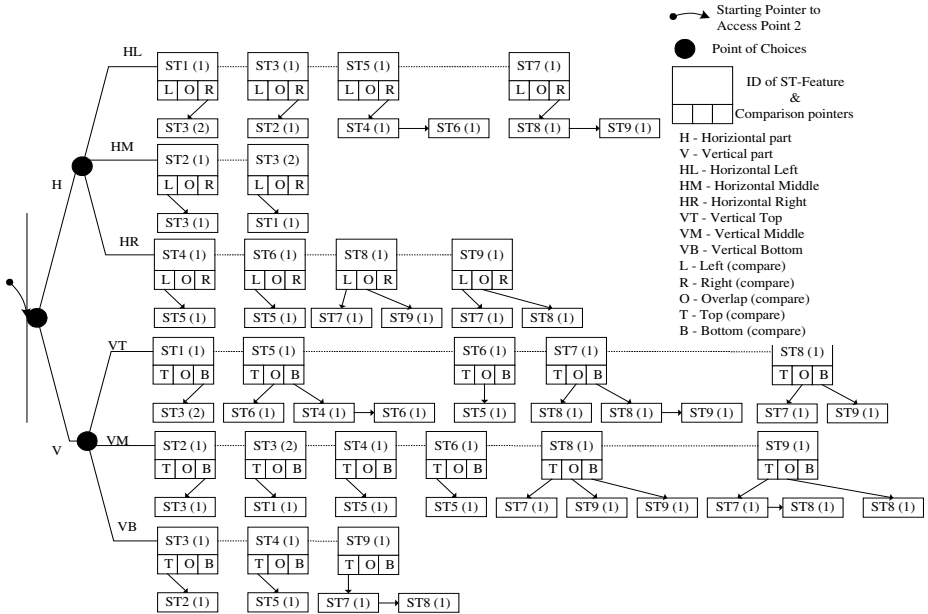


Fig. 10. An example of Access Point Two of the Query Indexing Mechanism (extended)

In order to allow user to specify more precisely and adequately spatial relationships among features, the indexing structure of Access Point Two can be extended to represent the binary or ternary relationships of Features, as shown in Fig. 10. For each node of Fig. 10, there are three kinds of pointers being extended to the structure. From the Horizontal part (H), the pointers are L, O, and R. From the Vertical part (V), the pointers are T, O, and B. L stands for relative left; R stands for relative right; T stands for relative top; B stands for relative bottom; and O stands for overlap. As shown in the Video frame interval 1 of Fig. 7, ST2(1) and ST3(1) occur in the same video frame interval. A user may want to know whether one feature is on the top of another; the query may be specified in CCM/ST as follows (x is a set of scene objects):

```
SELECT x FROM SCENE y
WHERE y HAS FEATURE Sun AND y HAS FEATURE Moon
AND F_INTERVAL(Sun) OVERLAP F_INTERVAL(Moon)
AND Sun = TOP(Sun, Moon);
```

To process this query, we need to search the Feature ‘Sun’ and the Feature ‘Moon’ from the Feature Tree. When these are found, check whether the Feature-Video association has linked to them and whether the same video object has linked to them. Also check whether they contain the spatio-temporal semantics, ST-Feature. After obtaining the IDs of the ST-Feature, we enter the Access Point One of the indexing mechanism (cf. Fig. 9), and get the “Vertical” pointers of Access Point Two. Now we search one of the Features from the Access Point Two and check whether the Feature is on the top/bottom of the other Feature by the pointers T or B (cf. Fig. 10). An alternative way is to search the Feature Sun and the Feature Moon from the Feature Tree. Then check whether they contain the spatio-temporal semantics, ST-Feature. After obtaining their IDs of ST-Feature, we enter the Access Point One of the indexing mechanism, and get the “Vertical” pointers of Access Point Two. Then search both Features from the Access Point Two and check whether they have relative positions. If they are found in the structure and have relative positions, it means they occur on the same video frame interval.[∞] Finally, check which one is on the top of the other. Conclusively, the spatial functions [CL1999] can be used to evaluate the queries of section 4.1(C) as desired.

4.2 On The Activity Model

As discussed in section 3.3, an Activity Model can be used to facilitate the annotation of significant activities for different application domains. Depending on the applications, the size of the model could be very large and hence impact the performance of the system. To solve this problem, there are two kinds of structure that could be used by each application, i.e. an importing structure, and an indexing structure of the Activity Model.

4.2.1 Importing Structure

The importing structure is a text file, which is analyzed and interpreted by the language processors. An example of an importing structure of a “Sports” domain is shown in Fig. 11. Note that the importing structure may evolve due to the user preference/perception change from time to time. It is thus necessary to store the structure separately for each application. Therefore, we suggest dividing the model into separate domains and importing domains into the system one at a time.

```
ACTIVITY="sports"
  EVENT="football"
    MOTION="handling"
      OBJECT="hand"
      OBJECT="football"
    MOTION="kicking"
      OBJECT="football"
      OBJECT="leg"
  EVENT="squash"
    MOTION="hitting"
      OBJECT="hand"
      OBJECT="racket"
      OBJECT="squash"
```

Fig. 11. A Text File Structure of a “Sports” domain of the Activity Model

4.2.2 Indexing Structure

In order to reduce the access time of the activity model, useful part of the importing structure of the activity model should be indexed. The indexing structure of the Activity Model is in the format of Fig. 12; an example is shown in Fig. 13.

[∞] For example, in the VM list of the Access Point Two of the indexing mechanism, ST2(1) has a pointer B pointed to ST3(1); also in the VB list, ST3(1) has a pointer T pointed to ST2(1).

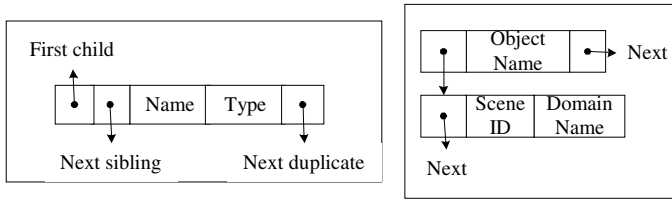


Fig. 12. An Indexing Structure used by the Activity Model

In Fig. 12, the left-hand side is to store the four level of the Activity Model. The right-hand side is to store the association of the Object level and the Scene, and is used to update the data of the left-hand side. Since Scene Order Tree nodes may be deleted, it is necessary to keep each node of the indexing structure updated and referenced.

For example, if a user only inputs “kicking football” into the Specification Language Processing Component, then the structure only stores (1) “sports” of the Activity level, (2) “football” of the Event level, (3) “kicking” of the Motion level, and (4) “football” and “leg” of the Object level. The indexing structure for this example is shown in Fig. 13.

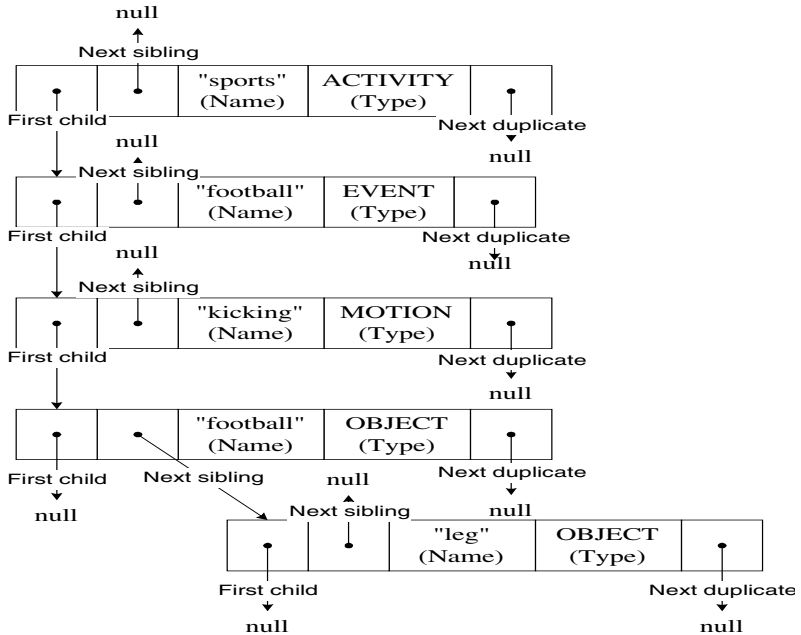


Fig. 13. An example of an Indexing Structure used by the Activity Model

4.3 User Interface & Implementation Environment

As part of our prototype, an end-user interface (EUI) has been developed which has the overall screen layout as shown in Fig. 14. CCM and CAROL have been developed using Microsoft Visual C++ and it uses an Object-oriented database engine (NeoAccess). The EUI can be divided into four different sections: the top left-handed section represents the Scene order tree and the top right-handed section represents the Feature index tree, the bottom left-handed section represents the Video player tree, and remaining section is used to display the video objects to the users. Through this EUI, users can perform dynamic creation, deletion and modification of scene, feature, and video player nodes, in addition to display of videos within

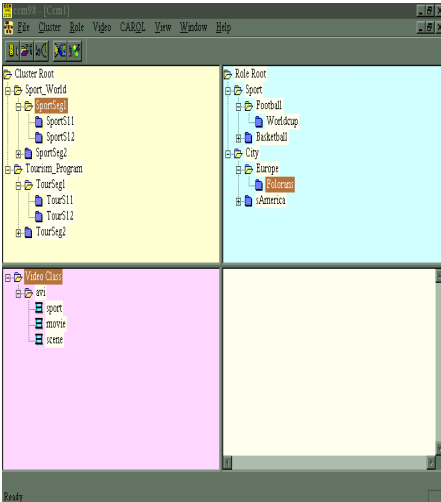


Fig. 15. Screen layout of our Video DBMS prototype

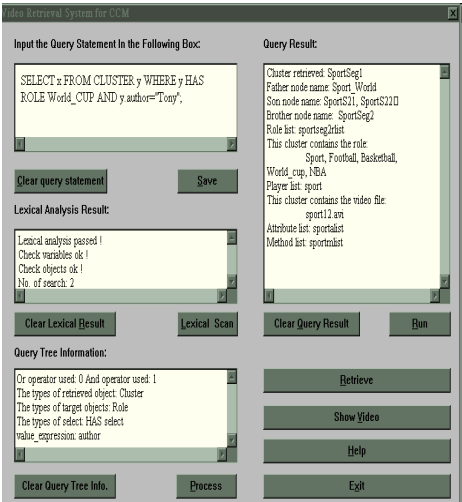


Fig. 14. Screen layout of CAROL

the interface. For users to issue queries using the query language CAROL, an interactive window is also provided as shown in Fig. 15. The query results can be shown and retrieved videos displayed upon user’s request.

5 Conclusions and future work

In this paper, we have described an important extension of an object-oriented video database system, based on the spatial and temporal semantics of video data. In addition, an enhancement of a query language, CAROL/ST has also been introduced, the development of which follows the development of CCM/ST since the latter utilizes the features provided by the former. The resultant language is able to accommodate more kinds of queries than its predecessor (CAROL), particularly from the perspectives of expressing spatial and temporal semantics.

To overcome the limitations of current computer vision and imaging processing techniques in extracting high-level (such as motion) semantics, a *Specification Language* is proposed to facilitate abstract level annotations by more sophisticated users (such as professional video editors and/or producers). The user interface receives from the professional users abstracted information such as event, action, and movement of significant objects contained in the video objects, and then converts the information from the abstracted level into primitive level. For example, “kicking football” is converted into “movement of football” and “movement of human”, and then converted into a “Football” feature with a ST-Feature coding [CL1999], and a Feature “Person” with a ST-Feature, plus a constraint/condition that the two ST-Features occupy the same frame intervals. Such primitive level of information is stored in the database, which can later be used by CAROL/ST in processing user queries. Furthermore, CCM/ST is enhanced to support the use of multi-level spatial information, so that user may choose different levels of ST-Features in the same application.

Admittedly, there are a number of issues that need to be addressed in our further research. In particular, scene ordering will be enhanced to support more temporal operators [CL1999, SHC1998]. An Object Composition Petri-Net (OCPN) [BF1998] is a good candidate to adopt for representing the temporal scenario of scenes. To add more temporal ordering functions [CL1999], aggregate functions [S1995], and more spatial operators [CL1999, VTS1998] may also be considered and possibly devised, which constitutes another interesting task for our subsequent research.

Acknowledgement

The authors would like to thank Hung Hang Fai (Andy) of CityU, for his contribution to the development of the specification language of the video database system.

References

- [BF1998] E. Bertino and E. Ferrari. Temporal Synchronization Models for Multimedia Data, *IEEE Transactions on Knowledge and Data Engineering*, Vol.10, No.4, July/August 1998.
- [CL1999] S. S. M. Chan and Q. Li. Facilitating Spatio-Temporal Operations in a Versatile Video Database System, *Proc. of 5th Int'l Workshop on Multimedia Information Systems (MIS'99)*, October 1999 (to appear).
- [HLLX1996] L. Huang, C.M. Lee, Q. Li and W. Xiong. An Experimental Video Database Management System Based on Advanced Object-Oriented Techniques. *IS&T/SPIE Conf. on Storage and Retrieval for Still Image and Video Database IV (Proceedings Vol.2670)*, pp.158-169, 1996.
- [IEEE1995] IEEE Computer. *Special issue on Finding the Right Image: Content-based Image Retrieval Systems*. Vol. 28, No.9, Dec. 1995.
- [KBG1989] W. Kim, E. Bertino, and J. F. Garza. Composite object revisited, *Proc. of ACM SIGMOD Intl. Conf. On Management of Data*, pp.337-347, 1989.
- [LH1995] Q. Li and L.S. Huang. A Dynamic Data Model for a Video Database Management System. *ACM Computing Surveys*, 27(4):602-606, 1995.
- [LL1995] Q. Li and C.M. Lee. Dynamic Object Clustering for Video Database Manipulations. *Proc. IFIP 2.6 Working Conf. on Visual Database Systems*, pp.125-137, Switzerland, 1995.
- [LL1996] Q. Li and K.M. Lam. Context-Sensitive and Context-Free Retrieval in a Video Database System. *Proc. 1st Int'l Workshop on Image Database and Multi Media Retrieval (IDB-MMS'96)*, pp.123-130, Amsterdam, 1996.
- [LLX1997] John C. M. Lee, Q. Li and W. Xiong. VIMS: A Video Information Management System, *Multimedia Tools and Applications*, January 1997, 4(1), 7-28.
- [LS1992] Q. Li and J.L. Smith. A Conceptual Model for Dynamic Clustering in Object Databases. *Proc. 18th Int'l Conf. on Very Large Data Bases (VLDB'92)*, pp.457-468, 1992.
- [LYC1998] Q. Li, Y. Yang and W.K. Chung. CAROL: Towards a Declarative Video Data Retrieval Language, *Proc. SPIE's Photonics China: Electronic Imaging and Multimedia Systems*, pp.69-78, Beijing, China, Sept 1998.
- [RL1997] E. Ryu and K. Lee. A Multimedia Query Language for Handling Multi-Structure Information, *Proc. of 5th Int'l Conference on Database System for Advanced Applications*, Melbourne, Australia, April 1-4, 1997.
- [S1995] R. T. Snodgrass. *The SQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.
- [S1998] R. Snodgrass. Special Series on Temporal Database, Part 1: Of Duplicates and Septuplets, *Database Programming & Design*, June 1998.
- [S1998-2] R. Snodgrass. Special Series on Temporal Database, Part 2: Querying Valid-Time State Tables, *Database Programming & Design*, July 1998.
- [SCZ1995] P. Scarponcini, D. C. St. Clair and G. W. Zobrist. An Inferencing Language for Automated Spatial Reasoning About Graphic Entities, *Proc. of 4th International Symposium, SSD'95*, Portland, ME, USA, August 1995, 259-278.
- [SHC1998] S. Y. W. Su, S. J. Hyun and H. H. M. Chen. Temporal Association Algebra: A Mathematical Foundation for Processing Object-Oriented Temporal Databases, *IEEE Transactions on Knowledge and Data Engineering*, Vol.10, No.3, 1998.
- [TCG1993] U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev and R. Snodgrass. *Temporal Databases: Theory, Design, And Implementation*, The Benjamin/Cummings Publishing Company, Inc, 1993.
- [VSS1998] V.S. Subrahmanian. *Principles of Multimedia Database Systems*, Morgan Kaufmann Publishers, Inc, 1998.
- [VTS1998] M. Vazirgiannis, Y. Theodoridis and T. Sellis. Spatio-temporal composition and indexing for large multimedia applications, *Multimedia Systems* 6: 284-298, 1998.

Entity Evolution in IsA Hierarchies

Antoni Olivé, Dolors Costal, and Maria-Ribera Sancho

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona (Catalonia)
e-mail: {olive, dolors, ribera}@lsi.upc.es

In memory of our colleague
and friend Anna Roselló

Abstract. In a conceptual schema that includes specializations and generalizations, and that allows multiple classification, an entity may be, at a given time, instance of one or more entity types. We call type configuration of an entity the set of entity types of which it is instance at some time. In this paper, we characterize the set of valid type configurations, taking into account the constraints defined by specializations and generalizations. We also analyze the problem of determining the valid evolution of the type configuration of entities in the context of *IsA* hierarchies. We describe the temporal features of entity types and the evolution constraints that influence entity evolution. In particular, we propose and define the specialization evolution constraints, which extend the classical distinction between static and dynamic entity types. Taking into account all these features and constraints, we characterize the set of valid type configuration transitions. In both cases, we tackle the problem for the general class of conceptual schemas, allowing multiple specialization, generalization and classification, as well as dynamic classification.

1 Introduction

Specialization and generalization are fundamental operations in conceptual modeling of information systems. Specialization consists in defining that an entity type (subtype) is a refinement of another one (supertype). The subtype is less general, or more specialized, than the supertype. Specialization establishes a relationship between entity types, usually called the *IsA* relationship. The resulting structure of the entity types in a conceptual schema is usually referred to as an *IsA* hierarchy. Most conceptual models allow an entity type to be subtype of two or more supertypes (multiple specialization). Generalization is the converse operation, and it consists in defining that an entity type is more general than a set of other entity types. Some conceptual models allow an entity type to be the generalization of two or more distinct sets of entity types (multiple generalization) [8, 20, 21, 28].

A specialization implies an inclusion constraint between the populations of the subtype and the supertype. Additional constraints on populations are usually defined in the context of generalizations, mainly the well-known union, disjointness and

partition constraints [11]. All these constraints are static, since they constrain the population of two or more entity types at the same time point (or state of the Information Base).

Many conceptual models allow entities to be instance of two or more entity types only if these entity types are related directly or indirectly by *IsA* relationships (single classification). Other conceptual models do not put this constraint, and allow entities to be instance of two or more entity types, even if these entity types are not related directly or indirectly by *IsA* relationships (multiple classification) [13]. For example, in multiple classification we could have an entity *Diana* instance of both *Manager* and *Engineer*. Handling multiple classification in single classification models require the introduction of auxiliary entity types and the use of multiple specialization. In the example, we should add an entity type such as *Manager&Engineer*, defined as a multiple specialization of *Manager* and *Engineer*, and then *Diana* could be (direct) instance of only *Manager&Engineer*.

Due to specialization and generalization, and to multiple classification, an entity may be instance of one or more entity types, at a given time. We call type configuration of an entity the set of entity types of which it is instance at some time. A set of entity types is a valid type configuration if there may exist an entity such that its type configuration at some time point is this set.

The first problem we address in this paper is the determination of the set of valid type configurations implied by a conceptual schema. In some cases, such determination is straightforward, but we have not found in the literature a formal characterization of that set in the general context of multiple classification, specialization and generalization. We give in this paper the conditions a type configuration must satisfy to be valid, taking into account the population constraints defined by specializations and generalizations. These conditions can be used to determine the set of valid type configurations.

Some conceptual models require type configurations of entities to be static; that is, the type configuration of an entity cannot change during the life span of that entity (static classification). Other conceptual models allow entities to change their type configuration (dynamic classification) [14]. For example, in dynamic classification we could have an entity *Albert* instance of *Man*, *Employee* and *Single* at some time, and instance of *Man*, *Retiree* and *Married* at some later time. Of course, not all evolutions are valid.

In this paper, we focus on conceptual models with dynamic classification and, in particular, we are interested in defining the constraints on the evolution of the entities. These constraints are domain knowledge relevant to the information system and, according to the completeness (or 100%) principle [10], they should be defined in the conceptual schema.

Some work has been already done in the characterization, at the conceptual level, of the valid evolution of entities. We mention here three directions that have been developed. First, there is the work on temporal characterization of individual entity types, which distinguishes among mutable, immutable, permanent, durable, event, etc. entity types [3, 16, 22, 25]. Such distinctions represent constraints on the life span of the entities in one entity type. A second direction is the work on the temporal characterization of entity types, in the context of *IsA* hierarchies, which distinguishes among static, dynamic, kind, role, etc. entity types [15, 28]. Such distinctions

represent constraints on the life span of the entities in two or more entity types. Finally, we mention the classical work on state-transition diagrams, which allows defining valid evolutions of entities in the context of a generalization or partition [6, 7, 12, 28].

The second problem we address in this paper is the determination of the valid type configuration transitions, that is, the valid evolution of type configurations of entities. We take into account previous work on temporal characterization of individual entity types, and on state-transition diagrams, and we define a new kind of evolution constraints that generalizes previous work on the temporal characterization in the context of IsA hierarchies. We give in this paper the conditions a type configuration transition must satisfy to be valid, taking into account all the above constraints, and for conceptual schemas with multiple classification, specialization and generalization, and dynamic classification.

The structure of the paper is as follows. Next section, which is based on [16], defines the temporal features of individual entity types that we consider. Section 3 deals with specializations, and defines the new evolution constraints that we propose, called specialization evolution constraints. Section 4 deals with partitions, and classifies them into three pure kinds. We also give here our own notation of the state-transition constraints, which will be called partition evolution constraints. Section 5 characterizes the set of valid type configurations, and Section 6 characterizes the set of valid type configuration transitions. Finally, in Section 7 we summarize our contribution, and point out future work.

2 Temporal Features of Entity Types

When we take a temporal view, we observe that entities and relationships are instances of their types at particular time points [1, 4, 23, 26]. We will assume that time points are expressed in a common base time unit (or *chronon* [25]), such as second or day. A time point represents an interval on the time-line.

We represent by $E(e, t)$ the fact that entity e is instance of entity type E at time t . For example, $Person(Albert, D1)$ means that *Albert* is instance of *Person* at time *D1* (day in this case). Note that $E(e, t)$ may hold either because e is instance of E at all possible subintervals of t , or because e is instance of E only at one or more subintervals of t . The *population* of E at t is defined as the set of entities that are instance of E at t . The *life span* of entity e in entity type E is the set of time points at which e is instance of E .

In this paper, we will use two temporal features of entity types that characterize the life span of their instances: durability and frequency. They were defined in [16], based on the work reported in [3, 5], and we reproduce their definition below for ease of reference.

2.1 Durability

The durability feature of an entity type has two possible values: instantaneous and durable. We say that entity type E is *instantaneous* if, for all e and t , $E(e,t)$ implies that:

- $E(e,t+1)$ cannot hold, or
- $E(e,t)$ holds only for some subintervals of t .

Otherwise, E is *durable*.

Most entity types are durable. For example, *Employee* is durable, because an entity P may be instance of *Employee* during two or more consecutive days T and $T+1$, and $Employee(P,T)$ may hold for all subintervals of day T . An example of instantaneous entity type may be *Birth* (of a person). A fact $Birth(B,T1)$, meaning that birth B happens at $T1$, only holds for a subinterval of $T1$, even if the base time unit is second.

If E is a durable entity type, a *classification interval* of e in E is a maximal set of consecutive time points of the life span of e in E . If E is instantaneous, then we define that there is a classification interval of e in E for each time point at which e is instance of E . Figure 1 shows several examples: $[t_4, t_6]$ is a classification interval of *Joan* in *Employee*; $[t_4, t_5]$ would not be a classification interval because it is not maximal.

The way by which an information system knows a classification interval of e in E depends on the durability of E . If E is durable, the system needs to be told only of the starting and ending times of the interval. If E is instantaneous, the system must be told explicitly of all times at which e is instance of E .

An entity e has one or more classification intervals in E . We denote by $Ci(e,E)$ the set of classification intervals of e in E . For example, in Figure 1 we have $Ci(B,Birth) = \{[t_3, t_3]\}$ and $Ci(Joan,Employee) = \{[t_1, t_2], [t_4, t_6]\}$. We will also need to use the predicate *IncludedIn*(t, ti) to indicate that time point t is included in the temporal interval ti . For instance, *IncludedIn*($t_4, [t_4, t_6]$) is true.

2.2 Frequency

The frequency feature of an entity type has also two possible values: single and intermittent. We say that entity type E is *single* if all entities that are instance of E at some time can only be instance of E during one, and only one, classification interval. Otherwise, E is *intermittent*.

Most entity types are single. For example, *Person* is single, if we assume that once a person ceases to exist, it cannot be reincarnated later (in the Information Base). An example of instantaneous and single entity type is *Birth*, since a birth can be instance of it only at a single time point. An example of durable and intermittent entity type could be *Employee*, if we assume that a person can be employed during several intervals. Finally, an example of instantaneous and intermittent entity type could be *Bride*. A fact $Bride(Diana, T1)$ means that entity *Diana* is bride at time $T1$. The same entity could be bride several times. Figure 1 shows some classification intervals of one example entity of each of the above types.

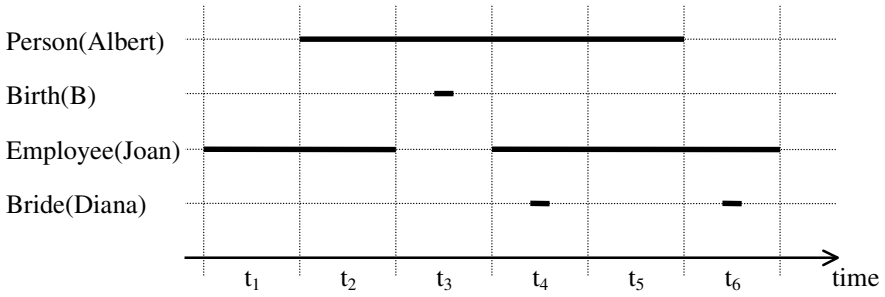


Fig. 1. Examples of classification intervals

3 Specializations and Specialization Evolution Constraints

A specialization is a relationship between two entity types E' and E , that we denote by $E' \text{ IsA } E$. E' is said to be a subtype of E , and E a supertype of E' . An entity type may be supertype of several entity types, and it may be a subtype of several entity types.

From a temporal point of view, the meaning of a specialization is that if an entity e is instance of E' at t , then it must also be instance of E at t . In the language of first order logic, this meaning is defined by the formula:

$$\forall e, t (E'(e, t) \rightarrow E(e, t))$$

The durability and frequency features of E' and E may be different. However, it is easy to see that there are two constraints:

- (1) Is E is instantaneous, then E' must also be instantaneous; and
- (2) if E is instantaneous and single, then E' must be instantaneous and single.

For example, *Bride IsA Person* would be acceptable, even if *Bride* is instantaneous and intermittent and *Person* is durable and single.

It is useful to assume, without loss in generality, the existence of a special entity type, that we call *Entity*. *Entity* is a supertype of all entity types defined in a schema. If an entity e is instance of *Entity* at time t , then it must also be instance of some other entity type at t . That is, there is no entity that it is only instance of *Entity*. Formally, if $E_1, \dots, E_n$ are the entity types defined in a schema, then:

$$\forall e, t (Entity(e, t) \leftrightarrow (E_1(e, t) \vee \dots \vee E_n(e, t)))$$

We define *Entity* as durable and intermittent.

An alternative, but equivalent, definition of specialization is in terms of type configurations. The type configuration of entity e at time t , denoted by $TC(e, t)$, is the set of entity types $TC(e, t) = \{E_i \mid e \text{ is instance of } E_i \text{ at } t\}$. Other names used sometimes for this set are „role set“ [24] and „context“ [15]. Note that *Entity* must be an element of all type configurations. In these terms, the meaning of a $E' \text{ IsA } E$ is:

$$\forall e, t (E' \in TC(e, t) \rightarrow E \in TC(e, t))$$

By itself, a specialization does not constraint the evolution of the type configuration of a given entity. For example, if we have *Man IsA Person* and $TC(Albert, 10) = \{Entity, Person, Man\}$, nothing prevents $TC(Albert, 11) = \{Entity, Person\}$. This

evolution would not be acceptable in this case, while it would be quite natural to have *Student* IsA *Person*, $TC(Albert, 10) = \{Entity, Person, Student\}$ and $TC(Albert, 11) = \{Entity, Person\}$.

In order to determine the valid evolutions of type configurations, we have defined three *specialization evolution constraints*, which characterize the possible evolution of the populations of the subtype and supertype in a specialization. We call them absolute static, relative static and dynamic. Specialization evolution constraints are a particular class of general dynamic constraints. Other evolution constraints will be defined later.

The absolute static constraint is more restrictive than the relative static one, which, in turn, is more restrictive than the dynamic one. In fact, the latter does not put any constraint on the evolution of the populations of the subtype and supertype. For each specialization, the designer should define which of the three constraints best corresponds to the evolution of the populations of the subtype and the supertype.

We say that E' is *absolute static* with respect to (wrt) E if there is a specialization $E' \text{ IsA } E$ and all entities that are instance of E' at some time cannot be, at some other previous or later time, instance of E without being also instance of E' . The formal definition is:

$$\forall e (\exists t E'(e, t) \rightarrow \forall t' (E(e, t') \rightarrow E'(e, t')))$$

which is equivalent to:

$$\forall e (\exists t E'(e, t) \rightarrow Ci(e, E') = Ci(e, E))$$

For example, *Man* is absolute static wrt *Person*: if *Albert* is instance of *Man* at time 10, then it is not possible for him to be instance of *Person* at some other time without being also instance of *Man*. The classification intervals of *Albert* in *Man* and in *Person* are the same. On the other hand, *Student* is not absolute static wrt *Person*.

We say that E' is *relative static* wrt E if there is a specialization $E' \text{ IsA } E$ and all entities that are instance of E' at some time, and have a classification interval ci in E , cannot be, at some other time included in ci , instance of E without being also instance of E' . Formally, E' is relative static with respect to E if:

$$\forall e (\exists t, ti E'(e, t) \wedge ti \in Ci(e, E) \wedge Included(t, ti) \rightarrow \forall t' (Included(t', ti) \rightarrow E'(e, t')))$$

For example, *Single* is relative static with respect to *Unmarried*. If *Albert* is instance of *Single* at time 10 then, by *Single* IsA *Unmarried*, he is also instance of *Unmarried* at the same time. Assume that a classification interval of *Albert* in *Unmarried* is $[10, 11]$. Then, *Albert* must be instance of *Single* during $[10, 11]$. At some later time, *Albert* may be instance of *Unmarried* without being instance of *Single* as, for example, if *Albert* is divorced. Figure 2 illustrates this evolution, showing that the classification intervals of *Albert* in *Single* are a subset of those of *Albert* in *Unmarried*.

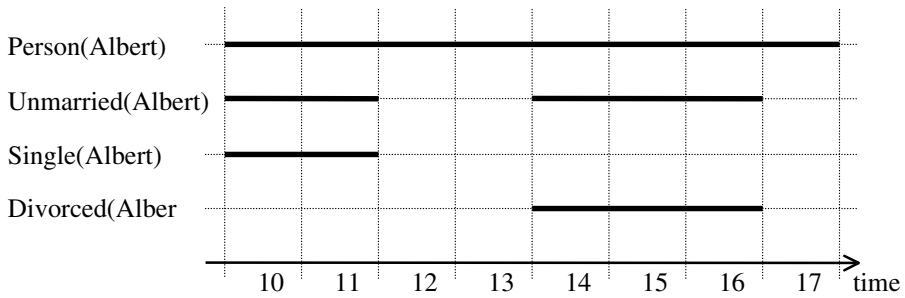


Fig. 2. Example evolution of type configurations

If E' is absolute static wrt E then it is also relative static. The converse is not true, in general. In the previous example, *Single* is not absolute static with respect to *Unmarried*. Thus, relative static is less restrictive than absolute static.

If E has single frequency then absolute static and relative static are equivalent. If E is instantaneous, then E' (that must also be instantaneous) is relative static with respect to E . For example, if *Birth* and *FamilyEvent* are both instantaneous and single, and *Birth* *IsA* *FamilyEvent* then *Birth* is absolute static wrt *FamilyEvent*.

Finally, we say that E' is *dynamic* wrt E if there is a specialization $E' \text{ IsA } E$. No additional constraints are imposed. For example, *Unmarried* is dynamic with respect to *Person*.

If E' is relative static wrt E , then it is also dynamic wrt E . The converse is not true, in general. In the previous example, *Unmarried* is not relative static wrt *Person*. Thus, dynamic is less restrictive than relative static.

It can be seen that if the constraint that holds between E'' and E' is α , and the constraint between E' and E is β (α, β = absolute static, relative static, dynamic), then the constraint that holds between E'' and E is the less restrictive of the two. For example, if we have:

Person absolute static wrt Entity.

Unmarried dynamic wrt Person.

then:

Unmarried dynamic wrt Entity.

If $E' \text{ IsA } E$ through two or more specialization paths, then the constraint between E' and E must be the same in all cases.

Note that our specialization evolution constraints are an extension of the classical distinction between static and dynamic entity types (or classes) [9, 28]. In our own terms, a static (dynamic) entity type is a durable entity type absolute static (dynamic) wrt *Entity*. As we have seen, our temporal features and evolution constraints offer much more possibilities, thus allowing, as we will see, a richer characterization of valid evolutions of type configurations.

4 Partitions and Partition Evolution Constraints

Generalization and specialization are two different viewpoints of the same *IsA* relationship, viewed from the supertype or from the subtype [17]. However, the term specialization tends to be used to mean a particular instance of the *IsA* relationship, such as *Man IsA Person*, while generalization means a set of *IsA* relationships with the same supertype. We adopt here this meaning, and denote by $E \text{ Gens } E_1, \dots, E_n$ ($n \geq 1$) the generalization of entity types $E_1, \dots, E_n$ to E . We write *Gens* as a shorthand for *Generalizes*. For example, *Person Gens Man, Woman*. There is an obvious relationship between *IsA* and *Gens*: if $E \text{ Gens } E_1, \dots, E_n$ then $E_1 \text{ IsA } E, \dots, E_n \text{ IsA } E$. Normally, the set of *IsA* included in a generalization are the result of some specialization criterion [19] or have the same classification principle [28] or discriminator [18].

The same entity type may be the supertype of several generalizations. For example, *Person Gens Man, Woman* and *Person Gens Child, Young, Adult*. On the other hand, an entity type may be a subtype in several generalizations. For example, *Person Gens Unemployed, Employee* and *TaxPayer Gens Employee, Company*.

Two well-known static constraints that apply to generalizations are the union and the disjointness constraints [7, 11, 12, 19, 27]. A generalization $E \text{ Gens } E_1, \dots, E_n$ is complete if every instance of E at t is also instance of at least one E_i at t ; otherwise, it is incomplete. A generalization $E \text{ Gens } E_1, \dots, E_n$ is disjoint if every instance of E at t is instance of at most one E_i at t ; otherwise, it is overlapping. A partition is a generalization that is both complete and disjoint. We write $P = E \text{ Partd } E_1, \dots, E_n$ to denote a partition of E , where *Partd* is a shorthand for *Partitioned*.

Every generalization can be transformed into a partition, through the use of auxiliary entity types and generalizations. The transformation is information-preserving, in the sense that the information content of the schema is not changed [2]. An incomplete generalization $E \text{ Gens } E_1, E_2$ can be transformed to a complete one by the introduction of an auxiliary entity type E_3 such that an entity is instance of E_3 at time t if it is instance of E and it is not instance of E_1 or E_2 , at time t . An overlapping generalization $E \text{ Gens } E_1, E_2$ can be transformed to a disjoint one by the introduction of three auxiliary entity types that we call $E_1 \text{ Non } E_2$, $E_1 E_2$ and $E_2 \text{ Non } E_1$, and the new partitions:

$$\begin{aligned} P_1 &= E \text{ Partd } E_1, E_2 \text{ Non } E_1; \\ P_2 &= E \text{ Partd } E_2, E_1 \text{ Non } E_2; \\ P_3 &= E_1 \text{ Partd } E_1 \text{ Non } E_2, E_1 E_2; \text{ and} \\ P_4 &= E_2 \text{ Partd } E_1 E_2, E_2 \text{ Non } E_1. \end{aligned}$$

Similar rules apply to generalizations with more than two subtypes.

Often it is easier to develop, analyze and reason about conceptual schemas, when only partitions are considered [6, 20, 28]. We have also found that the determination of valid configuration types and transitions is easier if we assume, without loss in generality, that all generalizations are partitions.

However, in some cases the definition of disjoint entity types by means of partitions may lead to some artificial entity types and partitions, difficult to understand. To avoid the need of such artificial constructs, we allow the explicit definition of disjointness constraints. In the above example, if we define that

$E1nonE2$ and $E2NonE1$ are disjoint, we do not need to define partitions P_3 and P_4 , nor the entity type $E1E2$.

In general, the specialization evolution constraint between a subtype and the supertype need not to be the same for all subtypes in the partition. For example, we might have:

Book Partd BookNotLoanable, BookOnLoan, BookAvailable

with *BookNotLoanable* absolute static wrt *Book*, and the other two dynamic. We say that a partition is *hybrid* if the above constraint is not the same for all subtypes. Otherwise, we say the partition is *pure*.

For the same reason as above, we find it convenient to assume that partitions are pure. Every hybrid partition can be transformed, also in an information-preserving way, into a pure one by introducing auxiliary entity types and partitions. Assume $E \text{ Partd } E_1, \dots, E_n$ is hybrid. The procedure of the transformation is as simple as:

1. If one or more subtypes $E_1, \dots, E_i$ are absolute static wrt the supertype, we generalize the other ones $E_{i+1}, \dots, E_n$ to a new entity type E' , which will be absolute static with respect to E . We define also a new partition $E' \text{ Partd } E_{i+1}, \dots, E_n$.

After the application of this step, the remaining hybrid partitions do not contain subtypes absolute static with respect to the supertype.

2. If one or more subtypes $E_1, \dots, E_i$ are relative static wrt the supertype, we generalize the other ones $E_{i+1}, \dots, E_n$ to a new entity type E' , which will be relative static wrt E . We define also a new partition $E' \text{ Partd } E_{i+1}, \dots, E_n$.

After the application of this step all partitions are pure.

Consider now an absolute static partition. By definition, whenever an entity is instance of the supertype it will also be instance of only one subtype, and it will always be instance of the same subtype.

The situation is a little bit different in a relative static partition, because we must take into account the classification intervals of entities in the supertype. Assume $E \text{ Partd } E_1, \dots, E_n$ is a relative static partition. Now, whenever an entity e is instance of E it will also be instance of only one subtype, and it will always be instance of the same subtype during a classification interval of e in E . The subtype may be different in different classification intervals.

In a dynamic partition, instances of the supertype must also be instance of only one subtype, but they can change the subtype at any time.

In relative static and dynamic partitions, it is often the case that there is an order in which entities can change subtype. For example, in *Person Partd Child, Young, Adult*. In such cases, the admissible transitions are usually defined by means of state diagrams [6, 7, 12, 28]. For our purposes, it is not necessary to take into account all elements that make up a state diagram, but only the *partition evolution constraints*, which define the allowed initial states, the valid transitions, and the allowed final states. The notation we use is as follows.

Let $P = E \text{ Partd } E_1, \dots, E_n$ be a relative static partition. The initial subtypes of P , denoted $IS(P)$, is a set of subtypes $IS(P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E must also be instance of some $E_i \in IS(P)$ in their first classification interval in E . The successor subtypes of E_i in P , denoted by $SS(E_i, P)$, is a set of subtypes $SS(E_i, P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E_i in a classification interval may be instance of some $E_j \in SS(E_i, P)$ in the next one. The final subtypes of P , denoted $FS(P)$, is a set of

subtypes $FS(P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E must also be instance of some $E_i \in FS(P)$ in their last classification interval in E .

For example, $P = \text{Unmarried Partd Single, Divorced, Widower}$ is a relative static partition, where:

$$\begin{aligned} IS(P) &= \{\text{Single}\} \\ SS(\text{Single}, P) &= \{\text{Divorced, Widower}\} \\ SS(\text{Divorced}, P) &= \{\text{Divorced, Widower}\} \\ SS(\text{Widower}, P) &= \{\text{Divorced, Widower}\} \\ FS(P) &= \{\text{Single, Divorced, Widower}\} \end{aligned}$$

Initial, successor and final subtypes are defined similarly for dynamic partitions. Let $P = E \text{ Partd } E_1, \dots, E_n$ be a dynamic partition. The initial subtypes of P , denoted $IS(P)$, is a set of subtypes $IS(P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E must also be instance of some $E_i \in IS(P)$ at the first time they are instance of E . The successor subtypes of E_i in P , denoted by $SS(E_i, P)$, is a set of subtypes $SS(E_i, P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E_i at t may be instance of some $E_j \in SS(E_i, P)$ at the next time they are instance of E . The final subtypes of P , denoted $FS(P)$, is a set of subtypes $FS(P) \subseteq \{E_1, \dots, E_n\}$ such that instances of E must also be instance of some $E_i \in FS(P)$ at the end of their last classification interval in E .

For example, $P = \text{Person Partd Child, Young, Adult}$ is a dynamic partition, where:

$$\begin{aligned} IS(P) &= \{\text{Child}\} \\ SS(\text{Child}, P) &= \{\text{Child, Young}\} \\ SS(\text{Young}, P) &= \{\text{Young, Adult}\} \\ SS(\text{Adult}, P) &= \{\text{Adult}\} \\ FS(P) &= \{\text{Child, Young, Adult}\} \end{aligned}$$

Our partition evolution constraints generalize the constraints implied by the concepts of kind and role in [15]. Using our own terms, if $P = E \text{ Partd } E_1, \dots, E_n$ then a durable entity type E_i ($i = 1, \dots, n$) can be seen as a kind if $SS(E_i, P) = \{E_i\}$ and as a role otherwise.

5 Valid Type Configurations

In the previous sections, we have defined the temporal features and constraints (static and evolution) that we consider in this paper. Now, we want to solve the following problems, which are important for conceptual schema validation and, during information systems design, for the determination of transaction preconditions:

- 1) Which are the possible type configurations of entities?.
- 2) When an entity is created, which are the type configurations it may have?.
- 3) Which are the possible type configuration transitions?.

In this section, we give a solution to the first two problems. For the first one, we only need to take into account the static constraints (union, disjointness). For the second, we have to consider also some partition evolution constraints. In the next section, we give a solution to the third problem, which needs to take into account all features and constraints.

As we have said before, the type configuration of entity e at time t , denoted by $TC(e, t)$, is the set of entity types $TC(e, t) = \{E_i \mid e \text{ is instance of } E_i \text{ at } t\}$. It is obvious

that, in general, not all combinations of entity types may be type configurations of some entity. We say that a set of entity types $VTC = \{E_1, \dots, E_n\}$ is a *valid type configuration* if there may exist an entity e and a time t such that $TC(e, t) = VTC$.

The empty set is always a valid type configuration. The type configuration of an entity e may be empty at some time t if e has been instance of some entity type at some previous time, but it is not instance of any entity type at t (although it may be instance of some of them later). We say that an entity is *suspended* at t if it has an empty type configuration at t , that is, $TC(e, t) = \emptyset$; otherwise, we say it is *active*. [22].

The determination of the set of valid type configurations implied by a conceptual schema is straightforward when all specializations and generalizations have been transformed into partitions. In effect, in this case a set of entity types VTC is a valid type configuration if it satisfies the following conditions:

1. If $E \in VTC$ and there is a partition $P = E \text{ Partd } E_1, \dots, E_n$, then one, and only one, $E_i \in VTC$, ($i = 1, \dots, n$).
2. If $E_i \in VTC$ and there is a partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_n$, then $E \in VTC$.
3. If $E_i, E_j \in VTC$ then there does not exist an explicit disjointness constraint between E_i and E_j .

The first two conditions are logical consequences of the concept of partition:

- (1) if an entity is instance of a supertype then it must be also instance of one, and only one, subtype; and
- (2) if an entity is instance of a subtype it must be also instance of the supertype.

The last condition is a logical consequence of the disjointness constraints.

Any search algorithm that searches (naively) all possible combinations of entity types and selects those that satisfy the above conditions may suffice to determine the set of valid type configurations. Of course, better algorithms are possible, but we do not discuss them here.

Figure 3 shows an example conceptual schema, which we will use in the rest of this paper. In the example, there are six pure partitions, numbered P_1 - P_6 . The type of each partition (*as*, absolute static; *rs*, relative static; *dy*, dynamic) is indicated next to the small triangle denoting the partition. There are also 13 entity types. Their durability (*d*, durable; *i*, instantaneous) and frequency (*s*, single; *i*, intermittent) is indicated next to the rectangle denoting the corresponding entity type. Note the auxiliary entity types *SingleNonBachelor* and *ManNonBachelor*. Their origin is the multiple specialization *Bachelor IsA Single, Man*, which is seen as two different incomplete generalizations: *Single Gens Bachelor* and *Man Gens Bachelor*. Transformation of these generalizations into partitions gives P_6 and P_5 , respectively. We also define an explicit disjointness constraint between *SingleNonBachelor* and *Man*.

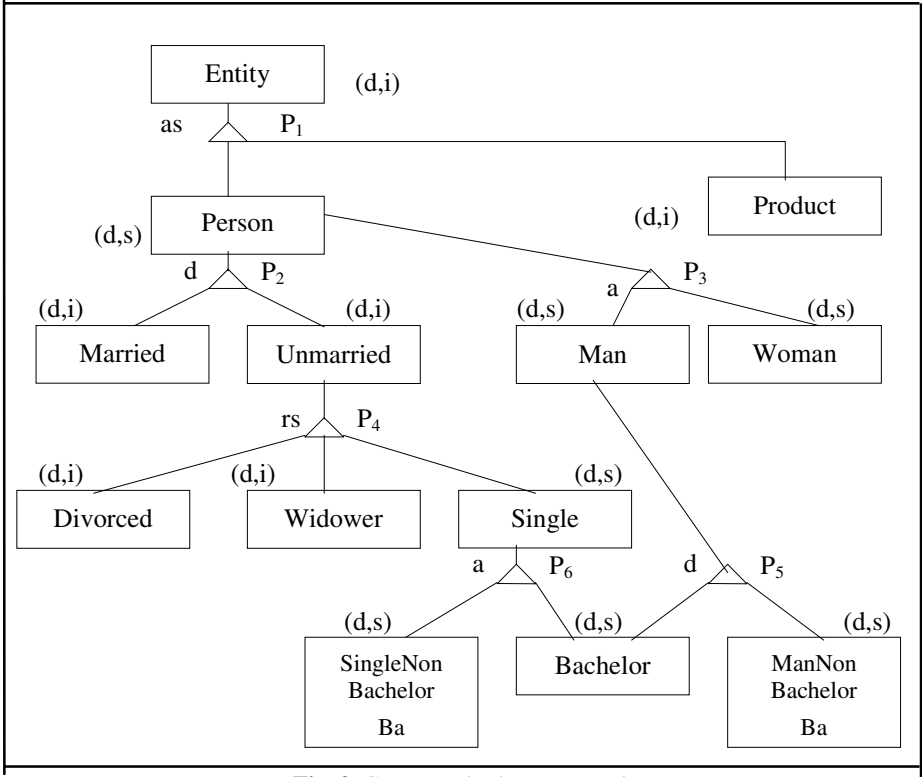


Fig. 3. Conceptual schema example

The conceptual schema of Figure 3 implies that there are 9 valid type configurations (in addition to the empty one), which we enumerate below:

$VTC_1 = \{\text{Entity, Product}\}$

$VTC_2 = \{\text{Entity, Person, Married, Man, ManNonBachelor}\}$

$VTC_3 = \{\text{Entity, Person, Married, Woman}\}$

$VTC_4 = \{\text{Entity, Person, Unmarried, Divorced, Man, ManNonBachelor}\}$

$VTC_5 = \{\text{Entity, Person, Unmarried, Divorced, Woman}\}$

$VTC_6 = \{\text{Entity, Person, Unmarried, Widower, Man, ManNonBachelor}\}$

$VTC_7 = \{\text{Entity, Person, Unmarried, Widower, Woman}\}$

$VTC_8 = \{\text{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman}\}$

$VTC_9 = \{\text{Entity, Person, Unmarried, Single, Bachelor, Man}\}$

(The auxiliary entity types could be removed from the above sets, if their presence is seen as confusing by developers or users. On the other hand, the above sets could be simplified by showing only the leaf entity types; in this way, for example, the last set could be shown by just $VTC_9 = \{\text{Bachelor}\}$).

Note that the type configuration:

$TC = \{\text{Entity, Person, Unmarried, Single, SingleNonBachelor, Man, ManNonBachelor}\}$

is not valid, due to the disjointness constraint (condition (3) above).

The initial type configuration of an entity is the type configuration the entity has when it is created, that is, when the information base knows it for the first time. Such configuration must be one of the above, but it must satisfy also the partition evolution constraints (initial subtypes). Formally, we say that a non-empty set of entity types $VITC = \{E_1, \dots, E_n\}$ is a *valid initial type configuration* if $VITC$ is a valid type configuration and:

1. If $E_i \in VITC$ and there is a partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_n$, then $E_i \in IS(P)$.

$IS(P_2) = \{\text{Unmarried}\}$
 $SS(\text{Unmarried}, P_2) = \{\text{Unmarried}, \text{Married}\}$
 $SS(\text{Married}, P_2) = \{\text{Unmarried}, \text{Married}\}$
 $FS(P_2) = \{\text{Unmarried}, \text{Married}\}$

 $IS(P_4) = \{\text{Single}\}$
 $SS(\text{Single}, P_4) = \{\text{Divorced}, \text{Widower}\}$
 $SS(\text{Divorced}, P_4) = \{\text{Divorced}, \text{Widower}\}$
 $SS(\text{Widower}, P_4) = \{\text{Divorced}, \text{Widower}\}$
 $FS(P_4) = \{\text{Single}, \text{Divorced}, \text{Widower}\}$

 $IS(P_5) = \{\text{Bachelor}\}$
 $SS(\text{Bachelor}, P_5) = \{\text{Bachelor}, \text{ManNonBachelor}\}$
 $SS(\text{ManNonBachelor}, P_5) = \{\text{ManNonBachelor}\}$
 $FS(P_5) = \{\text{Bachelor}, \text{ManNonBachelor}\}$

Fig. 4. Partition evolution constraints in the example of Figure 3.

Figure 4 shows the partition evolution constraints corresponding to the example of Figure 3. Applying the above condition, the set of valid initial type configurations is:

$VITC_1 = \{\text{Entity}, \text{Product}\} (= VTC_1)$

$VITC_2 = \{\text{Entity}, \text{Person}, \text{Unmarried}, \text{Single}, \text{SingleNonBachelor}, \text{Woman}\}$
 $(= VTC_8)$

$VITC_3 = \{\text{Entity}, \text{Person}, \text{Unmarried}, \text{Single}, \text{Bachelor}, \text{Man}\} (= VTC_9)$

6 Valid Type Configuration Transitions

When an entity is created, it adopts some type configuration, which must be a valid initial one. From that moment on, the entity may evolve and change its type configuration. At any time, the entity will have some type configuration, which may be empty if the entity is suspended. We define the *type configuration history* (or, for short, *history*) of an entity e at time t , $H(e, t)$, as the sequence of type configurations it has from its creation time until t :

$$H(e, t) = \langle TC(e, t_0), TC(e, t_0+1), \dots, TC(e, t-1), TC(e, t) \rangle$$

where t_0 is the creation time of e . All $TC(e, t_i)$ must be valid type configurations and, in particular, $TC(e, t_0)$ must be a valid initial type configuration. On the other hand, two consecutive type configurations may be the same, $TC(e, t_i-1) = TC(e, t_i)$, if e has not changed at t .

For example, assuming the conceptual schema of Figure 3 and, as base time unit, a month, the history of *Diana* on 05/99 could be:

$H(\text{Diana}, 05/99) =$
 $\langle \{\text{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman}\},$
 $\{\text{Entity, Person, Married, Woman}\},$
 $\{\text{Entity, Person, Married, Woman}\},$
 $\{\text{Entity, Person, Unmarried, Divorced, Woman}\} \rangle.$

A history $H(e, t)$ is *valid* if:

1. The life span of e in each of the entity types E_j included in some $TC(e, t_i)$ satisfies the temporal features of E_j , and
2. $H(e, t)$ satisfies the specialization and partition evolution constraints.

The history $H(e, t_0) = \langle TC(e, t_0) \rangle$ is always valid, provided that $TC(e, t_0)$ is a valid initial type configuration.

It is easier to characterize valid histories in terms of type configuration transitions (or, for short, transitions). A *transition* $Tr(e, t)$, $t > t_0$, is the pair $\langle H(e, t-1), TC(e, t) \rangle$, where $H(e, t-1)$ is the history of e at $t-1$, and $TC(e, t)$ is the type configuration of e at t . The evolution of an entity is a sequence of transitions. Once a transition takes place, the new history is:

$$H(e, t) = H(e, t-1) \oplus \langle TC(e, t) \rangle$$

where $\oplus$ denotes the sequence concatenation operator. We say that a transition is *valid* if both $H(e, t-1)$ and $H(e, t)$ are valid. It can be seen then that a history is valid if all transitions leading to it are valid.

In conceptual modeling and information systems design, we are not interested in particular histories and transitions, but in generic ones. A *valid generic history*, VGH , is a sequence of type configurations $VGH = \langle TC_0, \dots, TC_n \rangle$ such that there may exist an entity e created at time t_0 having the history $H(e, t_0+n) = VGH$. A *valid generic transition* is a pair $\langle \langle TC_0, \dots, TC_{n-1} \rangle, TC_n \rangle$ such that there may exist an entity e created at time t_0 having the history $H(e, t_0+n) = \langle TC_0, \dots, TC_{n-1}, TC_n \rangle$.

It is useful to distinguish five kinds of generic transitions $\langle \langle TC_0, \dots, TC_{n-1} \rangle, TC_n \rangle$:

1. **Reclassification:** when $TC_{n-1} \neq \emptyset$ and $TC_n \neq \emptyset$ and $TC_{n-1} \neq TC_n$.
2. **Active continuation:** when $TC_{n-1} = TC_n \neq \emptyset$.
3. **Suspension:** when $TC_{n-1} \neq \emptyset$ and $TC_n = \emptyset$.
4. **Suspension continuation:** when $TC_{n-1} = TC_n = \emptyset$.
5. **Reactivation:** when $TC_{n-1} = \emptyset$ and $TC_n \neq \emptyset$.

We give below the conditions that each kind of generic transition must satisfy to be valid.

6.1 Reclassification

Let $Tr = \langle \langle TC_0, \dots, TC_{n-1} \rangle, TC_n \rangle$ be a generic reclassification transition. Tr is valid if it satisfies the five following conditions:

- 1) If $E \in TC_{n-1}$ and E is instantaneous and single, then $E \notin TC_n$.
- 2) If there is an absolute or relative static partition $P = E \text{ Partd } E_1, \dots, E_p, \dots, E_m$ and
 - $E, E_i \in TC_{n-1}$, and
 - $E \in TC_n$

then $E_i \in TC_n$.

- 3) If there is a dynamic partition $P = E \text{ Partd } E_1, \dots, E_b, \dots, E_j, \dots, E_m$ and
- $E, E_i \in TC_{n-1}$, and
 - $E, E_j \in TC_n$

then $E_j \in SS(E_i, P)$.

- 4) If there is a relative static or dynamic partition $P = E \text{ Partd } E_1, \dots, E_b, \dots, E_j, \dots, E_m$ and
- there exists a k , $0 \leq k < n-1$, such that $E, E_i \in TC_k$, and
 - there does not exist a s , $k < s < n$, such that $E \in TC_s$, and
 - $E, E_j \in TC_n$

then $E_j \in SS(E_i, P)$.

- 5) If there is a relative static or dynamic partition $P = E \text{ Partd } E_1, \dots, E_b, \dots, E_m$ and
- $E, E_i \in TC_n$, and
 - there does not exist a k , $0 \leq k < n$, such that $E \in TC_k$

then $E_i \in IS(P)$.

The above conditions are logical consequences from the definitions of:

- (1) instantaneous and single.
- (2) absolute and relative static.
- (3) successor subtypes of dynamic partitions in the same classification interval.
- (4) successor subtypes of relative static and dynamic partitions, for different classification intervals.
- (5) initial subtypes of a partition.

For example, the reclassification:

Tr1 = <<{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman},
 {Entity, Person, Married, Woman}>,
 {Entity, Person, Unmarried, Divorced, Woman}>

would be valid.

An example of non-valid reclassification, violating condition (2) above, could be:

Tr2 = <<{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman},
 {Entity, Person, Married, Woman},
 {Entity, Person, Unmarried, Divorced, Woman}>,
 {Entity, Person, Unmarried, Widower, Woman}>

In this case, partition $P_4 = \text{Unmarried Partd Single, Widower, Divorced}$ is relative static and, therefore, an entity cannot change from *Divorced* to *Widower* in the same classification interval of *Unmarried*.

An example of non-valid reclassification, violating condition (4) above, could be:

Tr3 = <<{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman},
 {Entity, Person, Married, Woman}>,
 {Entity, Person, Unmarried, Single, SingleNonBachelor, Woman}>

In this case, the first type configuration includes *Unmarried* and *Single*, which are not present in the second one. As we have $SS(\text{Single}, P_4) = \{\text{Divorced}, \text{Widower}\}$, an entity cannot be *Single* again.

6.2 Active Continuation

Let $Tr = \langle \langle TC_0, \dots, TC_{n-1} \rangle, TC_n \rangle$ be a generic active continuation transition, with $TC_{n-1} = TC_n$. Tr is valid if it satisfies the two following conditions:

- 1) TC_{n-1} does not include any instantaneous and single entity type.
- 2) If there is a dynamic partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_m$ and
 - $E, E_i \in TC_{n-1}$
 - then $E_i \in SS(E_i, P)$.

For example, the active continuation:

$Tr4 = \langle \langle \dots, \{ \text{Entity, Person, Married, Man, ManNonBachelor} \} \rangle, \{ \text{Entity, Person, Married, Man, ManNonBachelor} \} \rangle$

would be valid.

6.3 Suspension

Let $Tr = \langle \langle TC_0, \dots, TC_{n-1} \rangle, \emptyset \rangle$ be a generic suspension transition. Tr is valid if it satisfies the following condition:

- 1) If there is a relative static or dynamic partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_n$ and
 - $E, E_i \in TC_{n-1}$ and
 - E is single
 - then $E_i \in FS(P)$.

If E is single, then it will not be possible to reactivate it later, for the same entity. Therefore, E_i must be one of the final subtypes of E in P .

For example, any suspension:

$Tr5 = \langle \langle \{ \text{Entity, Product} \}, \dots, \{ \text{Entity, Product} \} \rangle, \emptyset \rangle$

would be valid, because *Product* is intermittent, and therefore the above condition does not apply.

Note that for us an entity destruction is a suspension that is never reactivated.

6.4 Suspension Continuation

All generic suspension continuation transitions $Tr = \langle \langle TC_0, \dots, TC_{n-2}, \emptyset \rangle, \emptyset \rangle$ are valid.

6.5 Reactivation

Let $Tr = \langle \langle TC_0, \dots, TC_{n-2}, \emptyset \rangle, TC_n \rangle$ be a generic reactivation transition. Tr is valid if it satisfies the four following conditions:

- 1) If there exists a k , $0 \leq k < n-1$, such that $E \in TC_k$ and $E \in TC_n$ then E must be intermittent.
- 2) If there is an absolute static partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_n$ and
 - there exists a k , $0 \leq k < n-1$, such that $E, E_i \in TC_k$ and
 - $E \in TC_n$
 - then $E_i \in TC_n$.

3) If there is a relative static or dynamic partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_j, \dots, E_m$ and

- there exists a k , $0 \leq k < n-1$, such that $E, E_i \in TC_k$, and
- there does not exist a s , $k < s < n-1$, such that $E \in TC_s$, and
- $E, E_j \in TC_n$

then $E_j \in SS(E_i, P)$.

4) If there is a relative static or dynamic partition $P = E \text{ Partd } E_1, \dots, E_i, \dots, E_m$ and

- $E, E_i \in TC_n$, and
- there does not exist a k , $0 \leq k < n-1$, such that $E \in TC_k$

then $E_i \in IS(P)$.

The above conditions are logical consequences from the definitions of:

- (1) intermittent.
- (2) absolute static.
- (3) successor subtypes of relative static and dynamic partitions, for different classification intervals.
- (4) initial subtypes of a partition.

For example, the reactivation:

$\text{Tr6} = \langle \langle \{\text{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman}\}, \emptyset \rangle, \{\text{Entity, Person, Married, Woman}\} \rangle$

would be not be valid, since *Person* is not intermittent.

6.6 Valid Next Type Configurations

Using the above conditions, it is easy to compute all valid generic transitions from a given valid generic history. We call *valid next type configurations* $\text{VNTC}(\text{VGH})$ of a valid generic history VGH the set:

$\text{VNTC}(\text{VGH}) = \{TC_i \mid \langle \text{VGH}, TC_i \rangle \text{ is a valid generic transition} \}$

For example, if the valid generic history is

$\text{VGH} = \langle \{\text{Entity, Person, Unmarried, Single, SingleNonBachelor, Woman}\}, \{\text{Entity, Person, Married, Woman}\} \rangle$

then

$\text{VNTC}(\text{VGH}) =$
 $\{ \{\text{Entity, Person, Married, Woman}\},$
 $\{\text{Entity, Person, Unmarried, Divorced, Woman}\},$
 $\{\text{Entity, Person, Unmarried, Widower, Woman}\}, \emptyset \}$

7 Conclusions

We have analyzed the problem of determining the valid evolution of entities in the context of *IsA* hierarchies. We have seen that the problem has two facets: The type configurations an entity may have at a point in time and the possible evolutions of type configurations across time. The problem has been defined for the general class of

conceptual schemas, allowing multiple specialization, generalization and classification, as well as dynamic classification.

We have characterized the set of valid type configurations, taking into account the constraints defined by specializations and generalizations. We have shown that such characterization is simple when we transform generalizations into partitions.

We have described the temporal features and evolution constraints that influence entity evolution. In particular, we have proposed and defined the specialization evolution constraints, which extend the classical distinction between static and dynamic entity types. Taking into account these features and constraints, the paper characterizes the set of valid type configuration transitions.

Our work can be extended in at least three directions. First, it could be possible to define new temporal features or evolution constraints, or refine some of the proposed here. Such extension would require modifying accordingly the characterization of valid type configurations and transitions. Second, efficient algorithms could be developed for the generation of the set of valid type configurations and transitions implied by a given conceptual schema. This aspect has not been discussed in this paper, which has focused instead on the conditions those sets must satisfy. Finally, it would be interesting to analyze how to enforce the temporal features and evolution constraints in the structural events (preconditions of transactions). The basis for this study is the characterizations given in Sections 5 and 6 of this paper. See [5] for a similar work in a different context.

Acknowledgements

We would like to thank Juan Ramón López, Ernest Teniente, Toni Urpí and the anonymous referees for their useful comments. This work has been partially supported by CICYT program project TIC99-1048-C02-01.

References

1. Boman, M., Bubenko jr. J.A., Johannesson, P. and Wangler, B. "Conceptual Modelling", Prentice Hall, 1997.
2. Batini, C.; Ceri, S.; Navathe, S.B. "Conceptual Database Design. An Entity-Relationship Approach". The Benjamin/Cummings Pub. Co., 1992.
3. Bergamaschi, S. and Sartori, C. „Chrono: A conceptual design framework for temporal entities“. 17th Intl. Conf. on Conceptual Modeling (ER'98) - Singapore. LNCS 1507, pp. 35-50, 1998.
4. Bubenko, J.A.jr. "The Temporal Dimension in Information Modelling", In "Architecture and Models in Data Base Management Systems", North-Holland, pp. 93-113, 1977.
5. Costal, D.; Olivé, A. and Sancho, M-R. „Temporal Features of Class Populations and Attributes in Conceptual Models“. Proc. 16th. Intl. Conf. On Conceptual Modeling – ER'97, LNCS 1331, pp. 57-70, 1997.
6. de Champeaux, D.; Lea, D.; Faure, P. "Object-Oriented System Development." Addison-Wesley Pub. Co., 1993.

7. Embley, D.W.; Kurtz, B.D.; Woodfield, S.N. "Object-Oriented Systems Analysis." Yourdon Press., Prentice-Hall, Inc., 1992.
8. Halpin, T.A.; Proper, H.A. "Subtyping and polymorphism in object-role modelling", *Data & Knowledge Eng.*, 15, pp. 251-281, 1995.
9. Jungclaus, R.; Saake, G.; Hartmann, T.; Sernadas, C. "TROLL - A Language for Object-Oriented Specification of Information Systems", *ACM TOIS*, 14, 2, April, pp. 175-211, 1996.
10. ISO/TC97/SC5/WG3. "Concepts and Terminology for the Conceptual Schema and the Information Base", J.J. van Griethuysen (ed.), March 1982.
11. Lenzerini, M. "Covering and Disjointness Constraints in Type Networks", *Proc. ICDE87*, pp. 386-393, 1987.
12. Martin, J.; Odell, J. "Object-Oriented Methods: A Foundation". Prentice Hall, 1995.
13. Mostchnig-Pitrik, R.; Mylopoulos, J. "Classes and Instances". *Intl. J. of Intelligent and Cooperative Information Systems*, Vol.1, No.1, pp. 61-92, 1992.
14. Mylopoulos, J. "Information modeling in the time of the revolution", *Information Systems*, 23, 3/4, pp. 127-155, 1998.
15. Norrie, M.C.; Steiner, A.; Würzler, A. et al. "A Model for Classification Structures with Evolution Control". *Proc. 15th Intl Conf on Conceptual Modeling, LNCS 1157*, pp. 456-471, 1996.
16. Olivé, A. "Relationship Reification: A Temporal View". *Procs. of the CAiSE'99. LNCS 1626*, pp. 396-410, 1999.
17. Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. "Object-Oriented Modeling and Design". Prentice-Hall, Inc., 1991.
18. Rumbaugh, J.; Jacobson, I.; Booch, G. "The Unified Modeling Language Reference Manual". Addison-Wesley, 1999.
19. Saltor, F.; Castellanos, M.; Garcia, M. et al. Modelling "Specialization as BLOOM Semilattices". In "Information Modelling and Knowledge Bases", IOS Press, Vol. VI., 1995.
20. Smith, J.M.; Smith, D.C.P. "Database Abstractions: Aggregation and Generalization". *ACM TODS*, 2,2, pp. 105-133, 1977.
21. Snoeck, M.; Dedene, G. "Generalization/specialization and role in object oriented conceptual modeling", *Data and Knowledge Engineering*, 19, pp. 171-195, 1996.
22. Spaccapietra, S.; Parent, C.; Zimányi, E. "Modeling Time from a Conceptual Perspective". *CIKM*, pp. 432-440, 1998.
23. Steiner, A.; Norrie, M.C. "Temporal Object Role Modelling". *Proc. 9th CAISE'97, LNCS 1250*, pp. 243-258, 1997.
24. Su, J. "Dynamic Constraints and Object Migration". *Proc. 17th VLDB, Barcelona*, pp. 233-242, 1991.
25. Tansel, A.; Clifford, J.; Gadia, S. et al. "Temporal Databases: Theory, Design and Implementation". Benjamin/Cummings, 1993.
26. Theodoulidis, C.; Loucopoulos, P. "The Time Dimension in Conceptual Modelling". *Information Systems*, 16, 3, pp. 273-300, 1991.
27. Theodoulidis, C.; Wangler, B.; Loucopoulos, P. "The Entity-Relationship-Time Model". In "Conceptual Modeling, Databases and CASE. An Integrated View of Information Systems Development", John Wiley & Sons, pp. 87-115, 1992.
28. Wieringa, R.; de Jonge, W.; Spruit, P. "Using Dynamic Classes and Role Classes to Model Object Migration". *TPOS, Vol 1(1)*, pp. 61-83, 1995.

Temporal ER Modelling with Description Logics

Alessandro Artale¹ and Enrico Franconi²

¹ Department of Computation, UMIST, Manchester, UK,
artale@co.umist.ac.uk

² Department of Computer Science, University of Manchester, UK,
franconi@cs.man.ac.uk,
<http://www.cs.man.ac.uk/~franconi/>

Abstract. Recent efforts in the Conceptual Modelling community have been devoted to properly capturing time-varying information. Various temporally enhanced Entity-Relationship (ER) models have been proposed that are intended to model the temporal aspects of database conceptual schemas. This work gives a logical formalisation of the various properties that characterise and extend different temporal ER models which are found in literature. The formalisation we propose is based on Description Logics (DL), which have been proved useful for a logical reconstruction of the most popular conceptual data modelling formalisms. The proposed DL has the ability to express both enhanced temporal ER schemas and integrity constraints in the form of complex inclusion dependencies. Reasoning in the devised logic is decidable, thus allowing for automated deductions over the whole conceptual representation, which includes both the ER schema and the integrity constraints over it.

1 Introduction

Temporal Databases are databases that store historical information, i.e., essentially past and present data. Temporally enhanced Entity-Relationship (ER) models have been developed to conceptualise the temporal aspects of database schemas, namely *valid time* – when a fact holds, i.e., it is true in the representation of the world – and *transaction time* – which records the history of database states rather than the world history. In this paper we only consider the modelling of the validity time which is supported by almost all the temporal ER models – differently from the transaction time.

In the temporal ER community two different main modelling approaches have been devised to provide temporal support. The *implicit* approach hides the temporal dimension in the interpretation structure of the ER constructs. Thus, a temporal ER model does not include any new specific temporal construct with respect to a standard ER model. Each ER construct is always interpreted with a temporal semantics, so that instances of temporal entities or relationships are always potentially time-varying objects. The *explicit* approach, on the other hand, retains the non-temporal semantics for the conventional ER constructs, while adding new syntactical constructs for representing temporal entities and

relationships and their temporal interdependencies. The advantage of the explicit approach is the so called *upward compatibility*: the meanings of conventional (legacy) ER diagrams when used inside a temporal model remains unchanged. This is crucial, for example, in modelling data warehouses or federated databases, where sources may be a collection of both temporal and legacy databases.

A logical formalisation is introduced in this paper that can cover both the implicit and the explicit approaches. The idea is to provide a formalisation for implicit temporal ER models, enriched with the ability to express a powerful class of temporal integrity constraints. While instances of ER entities or relationships are potentially time-varying objects, integrity constraints can impose restrictions in the temporal validity of such objects. The formalisation is powerful enough that it is possible to explicitly state as integrity constraints the distinction between time-varying and snapshot (i.e., time invariant) constructs. In this way, an ER diagram may contain both temporal and non-temporal information, providing the ability to capture the explicit approach.

The formalisation presented in this paper is based on an expressive temporal Description Logic. Advantages of using Description Logics are their high expressivity combined with desirable computational properties – such as decidability, soundness and completeness of deduction procedures. The core is the Description Logic *ALCQI*, which is able to capture conventional ER models and has the ability to express a powerful class of inclusion dependency constraints. The core language is then extended with a tense logic. Temporal integrity constraints can be expressed in this combined logic, called *ALCQIT*. Reasoning in the whole framework is a decidable task, allowing for a complete calculus for temporal integrity constraints.

The paper is organised as follows. Section 2 introduces the temporally enhanced ER models, in both the implicit and explicit approaches. Section 3 first introduces the non-temporal *ALCQI* description logic, and then its extension by a standard tense modal logic. Section 4 will show how temporal Entity-Relationship schemas can be encoded into the temporal description logic, how additional temporal integrity constraints can be imposed on schemas, and how it is possible to reason in this framework. The final sections describe how integrity constraints can encode time-varying and snapshot constructs. Final remarks on possible extensions conclude the paper.

2 The Temporal ER Model

In this Section we introduce the temporally enhanced ER model and its semantics. We first consider standard ER diagrams.

Basic elements of ER schemas are *entities*, denoting a set of objects called *instances*, and *relationships*, denoting a set of *tuples* made by the instances of the different entities involved in the relationship. Since the same entity can be involved in the same relationship more than once, participation of entities in relationships is represented by means of *ER-roles*, to which a unique name is assigned. ER-roles can have *cardinality constraints* to limit the number instances

of an entity involved in the relationship. Both entities and relationships can have *attributes*, i.e., properties whose value belong to some predefined domain – e.g., **Integer**, **String**. Additionally, the considered ER model includes taxonomic relationships to state inclusion assertions between entities and between relationships. ER schemas are usually built using graphical tools. As showed in Figure 1, entities are represented by boxes, while diamonds are used for relationships. Attributes are ovals, while ER-roles are depicted by connecting relationships to each involved entity; in the figure, ER-roles do not have a name. Cardinality constraints on ER-roles are depicted by labelling the edges with the desired numeric constraints. An ISA relationship is shown with an oriented edge – from the more general to the more specific term – e.g., **Manager** ISA **Employee** in Figure 1.

So far we have considered a standard ER diagram, i.e., a diagram where no explicit temporal constructs appear. According to the implicit approach, a temporally enhanced ER diagram does not have any specific temporal construct, since it is intended that every construct has always a temporal interpretation. Thus, the syntax of the temporal model is the same as the standard one, and the temporal dimension is considered only at the semantical level.

The first-order semantics for the temporally enhanced ER model – in the implicit approach – is the standard and intuitive choice. We consider as a starting point the non temporal semantics introduced in [Calvanese *et al.*, 1998]. Interpretations of a temporal ER diagram $\mathcal{D}$ are those temporally dependent *database states* $\mathcal{B}(t) = (\Delta^{\mathcal{B}}, \cdot^{\mathcal{B}(t)})$ such that: each domain symbol D is mapped to a subset $D^{\mathcal{B}(t)}$ of the corresponding domain interpretation at time point t ; each entity symbol E is mapped to $E^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}}$ at time point t ; each attribute symbol A is mapped to $A^{\mathcal{B}(t)} : \Delta^{\mathcal{B}} \rightarrow \bigcup_i D_i^{\mathcal{B}(t)}$ at time point t ; each n -ary relationship, with both ER-roles and attributes, denotes a set of labelled n -tuples over $(\Delta^{\mathcal{B}} \bigcup_i D_i^{\mathcal{B}(t)})^n$ at time point t . A *legal database state* should satisfy the following conditions: If $E_1 \text{ ISA } E_2$, then $E_1^{\mathcal{B}(t)} \subseteq E_2^{\mathcal{B}(t)}$ for every t ; if A is an attribute of entity E with domain D , then $\forall e \in E^{\mathcal{B}(t)}. A^{\mathcal{B}(t)}(e) \in D^{\mathcal{B}(t)}$ for every t ; for each relationship R with $E_1, \dots, E_k$ entities involved by means of ER-roles $U_1, \dots, U_k$, and attributes $A_1, \dots, A_l$ with domains $D_1, \dots, D_l$, then every instance tuple of R at time t has the form $[U_1 : e_1, \dots, U_k : e_k, A_1 : a_1, \dots, A_l : a_l]$, where $e_i \in E_i^{\mathcal{B}(t)}$ and $a_j \in D_j^{\mathcal{B}(t)}$; for each minimum and maximum cardinality constraint, n, m respectively, in a ER-role U relating a relationship R with and entity E , then $n \leq \#\{r \in R^{\mathcal{B}(t)} \mid r[U] \in E^{\mathcal{B}(t)}\} \leq m$ for every t .

Let us consider the example ER diagram of Figure 1; this diagram is the running example considered in the survey paper [Gregersen and Jensen, 1999]. As we have noticed before, the implicit approach does not consider the temporal constructs related to the validity time of entities and relationships (see, e.g., the TEER model in [Elmasri and Navathe, 1994]). Thus, the example diagram should be modified, since there are some of those disallowed constructs. The *Profit* relationship becomes an attribute of the entity *Department*, the *Salary* relationship becomes an attribute of the entity *Employee*, the *Work-period* entity disappears, since it just denotes the validity time of the relationship *Works-for*.

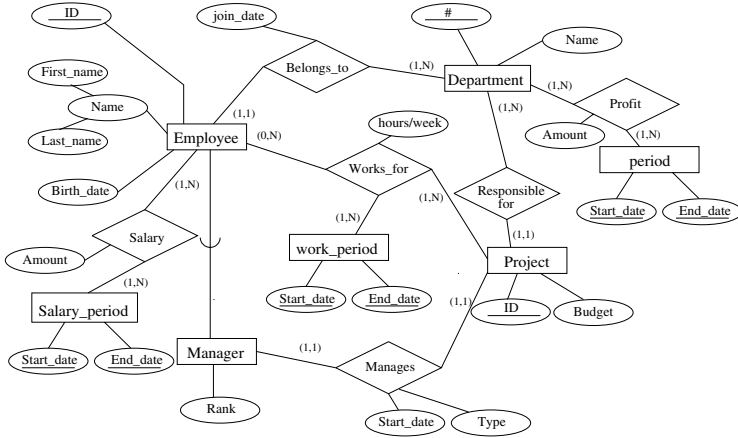


Fig. 1. The reference temporal ER diagram

The resulting diagram is such that *every* construct has its own validity time, conforming to the above temporal semantics.

We consider now an enhancement of the temporal ER model by means of integrity constraints. In this section just an informal introduction will be given; the exact characterisation of integrity constraints over a temporal ER diagram will be given in Section 4. The following integrity constraints may be imposed over the example ER diagram:

- managers are the only employees who do not work for a project (she/he just manages it);
- a manager becomes qualified after a period when she/he was just an employee¹.

These constraints can not be expressed directly by means of ER constructs; the logical formalisation presented in the following sections has been explicitly devised for expressing a large class of (temporal) constraints. The presence of the above constraints limits the number of legal database states, since not all the unconstrained databases conform to the newly introduced constraints. It is also easy to see that the legal database states conforming to the enriched schema, which includes both the ER diagram and the integrity constraints, also conform to the following constraints:

- for every project, there is at least an employee who is not a manager,
- each manager worked in a project before managing some (possibly different) project

¹ Let us assume that an entity called *Qualified* appears somewhere else in the ER diagram.

Please note that these deductions are not trivial, since from the ER schema the cardinality constraints do not impose that employees necessarily work in a project.

Explicit Approach

More complex and challenging becomes the case where an *explicit* approach to provide temporal support is adopted. In this case new constructs are added to represent the temporal dimension of the model. At the cost of adding new constructs, this approach has the advantage of preserving the atemporal meaning of conventional (legacy) ER schemas when embedded into temporal ER diagrams: this property is called *upward compatibility*. This crucial property is not realizable within the standard implicit temporal approach. Indeed, if the implicit approach has the attractive of leaving unchanged the original ER model, “this approach rules out the possibility of designing non-temporal databases or databases where some part of a database is non-temporal and the rest is temporal” [Gegersen and Jensen, 1999]. In particular, we suppose that both entities and relationships in a explicit temporal ER model can be either unmarked, in what case they are considered *snapshot* constructs (i.e., each of their instances has a global lifetime, as in the case they derive from a legacy diagram), or explicitly *temporary* marked (i.e., each of their instances has a temporary lifetime).

3 The Temporal Description Logic

We introduce very briefly in this section the *ALCQIT* temporal DL, which is obtained by combining a standard tense logic and the standard non-temporal *ALCQI* DL [Calvanese *et al.*, 1999].

The basic types of the DL are *concepts*, *roles*, and *features*. According to the syntax rules at the left of Figure 2, *ALCQI* *concepts* (denoted by the letters C and D) are built out of *primitive concepts* (denoted by the letter A), *roles* (denoted by the letter R, S), and *primitive features* (denoted by the letter f); roles are built out of *primitive roles* (denoted by the letter P) and *primitive features*.

We define the *meaning* of concepts as sets of individuals and the meaning of roles as sets of pairs of individuals. A temporal structure $\mathcal{T} = (\mathcal{P}, <)$ is assumed, where $\mathcal{P}$ is a set of time points and $<$ is a strict linear order on $\mathcal{P}$. Formally, an *ALCQIT* *temporal interpretation* over $\mathcal{T}$ is a triple $\mathcal{I} \doteq \langle \mathcal{T}, \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}(t)} \rangle$, consisting of a set $\Delta^{\mathcal{I}}$ of individuals (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}(t)}$ (the *interpretation function* of $\mathcal{I}$) mapping, for each $t \in \mathcal{P}$, every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every feature to a partial function from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{I}}$, such that the equations at the right of Figure 2 are satisfied.

A *knowledge base* is a finite set Σ of *terminological axioms* of the form $C \sqsubseteq D$. An interpretation $\mathcal{I}$ over a temporal structure $\mathcal{T} = (\mathcal{P}, <)$ satisfies a terminological axiom $C \sqsubseteq D$ if $C^{\mathcal{I}(t)} \subseteq D^{\mathcal{I}(t)}$ for every $t \in \mathcal{P}$. A knowledge base Σ is *satisfiable* in the temporal structure $\mathcal{T}$ if there is a temporal interpretation $\mathcal{I}$

$C, D \rightarrow$	$A \mid$	
	$\top \mid$	$\Delta^{\mathcal{I}}$
	$\perp \mid$	$\emptyset$
	$\neg C \mid$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}(t)}$
	$C \sqcap D \mid$	$C^{\mathcal{I}(t)} \cap D^{\mathcal{I}(t)}$
	$C \sqcup D \mid$	$C^{\mathcal{I}(t)} \cup D^{\mathcal{I}(t)}$
	$\forall R. C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \forall j. R^{\mathcal{I}(t)}(i, j) \Rightarrow C^{\mathcal{I}(t)}(j)\}$
	$\exists R. C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists j. R^{\mathcal{I}(t)}(i, j) \wedge C^{\mathcal{I}(t)}(j)\}$
	$f \uparrow \mid$	$\Delta^{\mathcal{I}} \setminus \text{dom } f^{\mathcal{I}(t)}$
	$f : C \mid$	$\{i \in \text{dom } f^{\mathcal{I}(t)} \mid C^{\mathcal{I}(t)}(f^{\mathcal{I}(t)}(i))\}$
	$\geq n R. C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \#\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}(t)}(i, j) \wedge C^{\mathcal{I}(t)}(j)\} \geq n\}$
	$\leq n R. C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \#\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}(t)}(i, j) \wedge C^{\mathcal{I}(t)}(j)\} \leq n\}$
	$CU D \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists v. v > t \wedge D^{\mathcal{I}(v)}(i) \wedge \forall w. (t < w < v) \rightarrow C^{\mathcal{I}(w)}(i)\}$
	$CS D \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists v. v < t \wedge D^{\mathcal{I}(v)}(i) \wedge \forall w. (v < w < t) \rightarrow C^{\mathcal{I}(w)}(i)\}$
	$\diamond^+ C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists v. v > t \wedge C^{\mathcal{I}(v)}(i)\}$
	$\diamond^- C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists v. v < t \wedge C^{\mathcal{I}(v)}(i)\}$
	$\Box^+ C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \forall v. v > t \rightarrow C^{\mathcal{I}(v)}(i)\}$
	$\Box^- C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \forall v. v < t \rightarrow C^{\mathcal{I}(v)}(i)\}$
$R, S \rightarrow$	$P \mid$	
	$f \mid$	
	$R^{-1} \mid$	$\{(i, j) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}(t)}(j, i)\}$
	$R _C \mid$	$R^{\mathcal{I}(t)} \cap (\Delta^{\mathcal{I}} \times C^{\mathcal{I}(t)})$
	$R \circ S$	$R^{\mathcal{I}(t)} \circ S^{\mathcal{I}(t)}$

Fig. 2. $\mathcal{ALCQIT}$ and its semantics.

over $\mathcal{T}$ which satisfies every axiom in Σ ; in this case $\mathcal{I}$ is called a *model* over $\mathcal{T}$ of Σ . Checking for KB satisfiability is deciding whether there is at least one model for the knowledge base. Σ *logically implies* an axiom $C \sqsubseteq D$ in the temporal structure $\mathcal{T}$ (written $\Sigma \models C \sqsubseteq D$) if $C \sqsubseteq D$ is satisfied by every model over $\mathcal{T}$ of Σ . In this latter case, the concept C is said to be *subsumed* by the concept D in the knowledge base Σ and the temporal structure $\mathcal{T}$. Concept subsumption can be reduced to concept satisfiability since C is subsumed by D in Σ if and only if $(C \sqcap \neg D)$ is unsatisfiable in Σ .

Reasoning in $\mathcal{ALCQI}$ (i.e., deciding knowledge base satisfiability and logical implication) is decidable, and it has been proven to be an EXPTIME-complete problem [Calvanese *et al.*, 1999]. The tense-logical extension of $\mathcal{ALCQI}$ has been inspired by the works of [Schild, 1993; Artale and Franconi, 1998; Artale and Franconi, 1999; Wolter and Zakharyashev, 1998b]. The following theorem states that reasoning in $\mathcal{ALCQIT}$ is decidable:

Theorem 1 (Decidability of $\mathcal{ALCQIT}$). *The problems of checking knowledge base satisfiability and logical implication are decidable for $\mathcal{ALCQIT}$ over a linear, unbounded, and discrete temporal structure (like the natural numbers).*

The proof is based on a reduction to the decidable language introduced in [Wolter and Zakharyashev, 1998a]. The exact computational complexity of reasoning in $\mathcal{ALCQIT}$ is still unknown; the lower bound is EXPTIME-hard.

As an example let us consider the axiom describing the situation in which any living mortal has mortal parents, has only living children (if any), remains alive until it will die, and at some point in the past was born:

$$\begin{aligned} \text{Mortal} \sqcap \text{LivingBeing} \sqsubseteq & \text{has-father} : \text{Mortal} \sqcap \text{has-mother} : \text{Mortal} \sqcap \\ & (\forall \text{has-father}^{-1}. (\text{Mortal} \sqcap \text{LivingBeing}) \sqcup \\ & \quad \forall \text{has-mother}^{-1}. (\text{Mortal} \sqcap \text{LivingBeing})) \sqcap \\ & \text{LivingBeing} \mathcal{U} \sqcap^+ \neg \text{LivingBeing} \sqcap \\ & \text{LivingBeing} \mathcal{S} \sqcap^- \neg \text{LivingBeing} \end{aligned}$$

4 Encoding the Implicit Temporal ER Model

Now, it is shown how an ER schema with implicit representation of time can be expressed in a $\mathcal{ALCQIT}$ knowledge base whose models correspond with legal database states of the ER schema – allowing for reasoning services such as satisfiability of a schema or the computation of logically implied integrity constraints. It is important to emphasise the fact that in this approach the integrity constraints are part of the schema, and that reasoning is carried on by taking in complete account all the information contained in the schema.

Let us first consider the translation between an ER diagram (without considering the integrity constraints) and an $\mathcal{ALCQIT}$ knowledge base as follows.

Definition 1 (Translation). *An ER diagram $\mathcal{D}$ is translated into a corresponding knowledge base Σ where each domain, entity or relationship symbol corresponds to an atomic concept, and each attribute or ER-role symbol corresponds to an atomic feature, in such a way that the following axioms hold:*

- For each ISA link between two entities E, F (resp. two relationships R, S) in $\mathcal{D}$, add to Σ the terminological axiom:
 $E \sqsubseteq F$ (resp. $R \sqsubseteq S$)
- For each attribute A in $\mathcal{D}$ with domain D of an entity E (resp. of a relationship R), add to Σ the terminological axiom:
 $E \sqsubseteq A : D$ (resp. $R \sqsubseteq A : D$)
- For each relationship R in $\mathcal{D}$ relating n entities $E_1 \dots E_n$ by means of the ER-roles² $P_{E_1}^R \dots P_{E_n}^R$, add to Σ the terminological axiom:
 $R \sqsubseteq (P_{E_1}^R : E_1) \sqcap \dots \sqcap (P_{E_n}^R : E_n)$
- For each minimum cardinality constraint $n \neq 0$ in a ER-role P_E^R in $\mathcal{D}$ relating a relationship R with and entity E , add to Σ the terminological axiom:
 $E \sqsubseteq \geq n (P_E^R)^{-1}. R$

² We assume that a unique name is given within a relationship to each ER-role, representing a specific participation of an entity in the relationship.

- For each maximum cardinality constraint $n \neq \infty$ in a ER-role P_E^R in $\mathcal{D}$ relating a relationship R with an entity E , add to Σ the terminological axiom:

$$E \sqsubseteq \leq n (P_E^R)^{-1} . R$$
□

From the definition 1 it is clear that n -ary relationships are *reified* in the translated description logic knowledge base, i.e., they become concepts with n special feature names – the ER-roles – denoting the n arguments of the n -ary relationship. Moreover, note that *ALCQIT* temporal operators do not appear in the translation, since we are in the implicit approach.

Temporal integrity constraints are expressed by means of additional terminological axioms in Σ . Recall that a terminological axiom states an inclusion between concepts. Thus, an integrity constraint is any kind of inclusion dependency expressible in the full temporal DL *ALCQIT*.

Definition 2 (Inclusion Dependencies). *An integrity constraint for an ER diagram $\mathcal{D}$ is any inclusion dependency which can be expressed in the corresponding knowledge base Σ by means of a terminological axiom of the kind $C \sqsubseteq D$, where atomic concepts appearing in C, D correspond to domain, entity or relationship symbols in $\mathcal{D}$ and atomic features appearing in C, D correspond to ER-role symbols in $\mathcal{D}$.*

Based on the results of [Calvanese *et al.*, 1994; Calvanese *et al.*, 1998], we have proved that the translation is correct, in the sense that there is a precise correspondence between legal database states of $\mathcal{D}$ and models of the derived knowledge base Σ . The existence of this correspondence is such that, whenever the problem of checking an ER schema against a property has a specific solution, then the corresponding reasoning problem in the DL has a corresponding solution, and vice-versa. Thus, it is possible to exploit standard reasoning procedures in the DL for checking properties of the ER schema. The reasoning problems we are mostly interested in are *consistency* of a ER schema – which is mapped to a satisfiability problem in the corresponding DL knowledge base – and *logical implication* within a ER schema – which is mapped to a logical implication problem in the corresponding DL knowledge base.

The proof is based by establishing the existence of two mappings from legal database states of $\mathcal{D}$ to models of Σ and vice-versa. Informally speaking, the mapping is possible since the temporal semantics of *ALCQIT* is such that every object is given a temporal interpretation, the same way a validity time is always associated to every tuple (or instance of entity) in the implicit approach. The existence of the mappings ensures that, whenever Σ is satisfiable, it is possible to build a corresponding non-empty legal database state, and vice-versa.

As a final remark, it should be noted that the high expressivity of DL constructs can capture an extended version of the basic ER model, which includes not only taxonomic relationships, but also arbitrary boolean constructs to represent so called generalized hierarchies with disjoint unions; entity definitions by means of either necessary or sufficient conditions or both, and integrity constraints expressed by means of generalised axioms [Calvanese *et al.*, 1998].

Example

Let us consider the example introduced in Section 2. We first translate the fragment of the ER diagram (Figure 1) involving the entities *Project*, *Employee*, *Manager* and the relationship *Works-for* in the description logic knowledge base Σ_{ER} :

WORKS-FOR $\sqsubseteq$ has-prj : Project $\sqcap$ has-emp : Employee
 Project $\sqsubseteq \exists \text{has-prj}^{-1}. \text{WORKS-FOR}$
 Manager $\sqsubseteq$ Employee

We then encode the integrity constraints, which are expressed by means of terminological axioms in a knowledge base Σ_{IC} :

- Managers are the only employees who do not work for a project:
 Manager $\sqsubseteq \forall \text{has-emp}^{-1}. \neg \text{WORKS-FOR}$
 $\forall \text{has-emp}^{-1}. \neg \text{WORKS-FOR} \sqsubseteq \text{Manager}$
 The constraints state that managers are exactly employees which are not involved in the *Works-for* relation.
- A manager becomes qualified after a period when she/he was just an employee:
 Manager $\sqsubseteq \text{Qualified } S (\text{Employee} \sqcap \neg \text{Manager})$
 The constraint states that all managers are qualified after they have been at the same time employees and not managers.

It turns out that the following integrity constraints are logically implied from $\Sigma_{ER} \cup \Sigma_{IC}$:

- For every project, there is at least an employee who is not a manager:
 $\Sigma_{ER} \cup \Sigma_{IC} \models \text{Project} \sqsubseteq \exists (\text{has-prj}^{-1} \circ \text{has-emp}). \neg \text{Manager}$
 The constraint states that every project is such that there exists somebody working for it who is not a manager.
- A manager worked in a project before managing some (possibly different) project:
 $\Sigma_{ER} \cup \Sigma_{IC} \models \text{Manager} \sqsubseteq \diamond^{-} \exists (\text{has-emp}^{-1} \circ \text{has-prj}). \text{Project}$
 The constraint states that every manager should have worked some time in the past for a project.

Moreover, if we change in Σ_{ER} the minimum cardinality of the participation of employees to the *Works-for* relationship to one (i.e., we make it a mandatory participation):

Employee $\sqsubseteq \exists \text{has-emp}^{-1}. \text{WORKS-FOR}$

then, even if Σ_{ER} is satisfiable, $\Sigma_{ER} \cup \Sigma_{IC}$ is an unsatisfiable knowledge base, because of the first integrity constraint. For the abovementioned theorem, no legal DB state exists for the ER schema including the constraints.

5 Encoding the Explicit Temporal ER Model

In the following it will be showed how the proposed formalisation can encode explicit temporal ER models by simply imposing specific constraints defining snapshot and temporary constructs, thus maintaining the required upward compatibility.

5.1 Snapshot Vs. Temporary Entities

The description logic *ALCQIT* is able to capture explicit temporal ER models by first applying the translation given in the previous Section, and then adding precise axioms to distinguish between snapshot and temporal constructs. In the following, axioms for entities are illustrated. In the next Section, the analogous for relationships will be showed.

A snapshot entity is axiomatised by the following constraint:

$$E \sqsubseteq (\Box^+ E) \sqcap (\Box^- E) \quad (\textit{Snapshot axiom})$$

expressing that whenever the entity is true it is necessarily true in every past and future time point. Indeed, instances of snapshot entities have necessarily a global lifetime. On the other hand, a temporary entity is axiomatised by the following constraint:

$$E \sqsubseteq (\Diamond^+ \neg E) \sqcup (\Diamond^- \neg E) \quad (\textit{Temporary axiom})$$

asserting that there must be a past or future time point where the entity does not hold. Indeed, instances of temporary entities have necessarily a limited lifetime.

With respect to our running example, the **Department** entity may be considered a snapshot entity, since it is unlikely that the organisational structure of an enterprise changes in time, while the entity **Manager** may be considered a temporary entity, since managers may change in time.

Using the reasoning capabilities of *ALCQIT* it is possible to support the database designer to discover relevant schema properties. As an example of the logical implications holding in a diagram making use of both snapshot and temporary entities, let us consider the interaction between entities via ISA links. Let us suppose that there is an ISA link between a snapshot entity E_1 and a temporary entity E_2 . This temporal ER diagram is translated into the following unsatisfiable knowledge base:

$$E_1 \sqsubseteq (\Box^+ E_1) \sqcap (\Box^- E_1)$$

$$E_2 \sqsubseteq (\Diamond^+ \neg E_2) \sqcup (\Diamond^- \neg E_2)$$

$$E_1 \sqsubseteq E_2$$

Thus, a snapshot entity can not be a subclass of a temporary entity, this is true also whenever such a kind of taxonomic relation is derived in the temporal ER model. This can be formally explained by observing that if this ISA relationship would be valid there will be an instance, say a , such that a is of type E_1 and

E_2 at a certain point t_0 – let represent this fact by means of the following set notation: $\{a : E_1, a : E_2\}_{t_0}$. As specified by the temporary axiom for E_2 , there must be a time point, say t_1 , such that a is not of type $E_2 - \{a : \neg E_2\}_{t_1}$. On the other hand, since E_1 is a snapshot entity, a is of kind E_1 at all time points, and in particular at time $t_1 - \{a : \neg E_2, a : E_1\}_{t_1}$. Due to the subclass relationship, this would imply that a is of type E_2 at $t_1 - \{a : \neg E_2, a : E_1, a : E_2\}_{t_1}$. Then, both a is of type E_2 and a is *not* of type E_2 holds at t_1 , which is a contradiction.

From these considerations it is easy to understand why the following implications hold:

$$\begin{aligned} \{E_2 \sqsubseteq (\Diamond^+ \neg E_2) \sqcup (\Diamond^- \neg E_2), E_1 \sqsubseteq E_2\} &\models E_1 \sqsubseteq (\Diamond^+ \neg E_1) \sqcup (\Diamond^- \neg E_1) \\ \{E_1 \sqsubseteq (\Box^+ E_1) \cap (\Box^- E_1), E_1 \sqsubseteq E_2\} &\models E_2 \sqsubseteq (\Box^+ E_2) \cap (\Box^- E_2) \end{aligned}$$

i.e., necessarily, every subclass of a temporary entity must be temporary; and a superclass of a snapshot entity must be a snapshot entity. Conversely, nothing can be said with respect to subclasses of snapshot entities. For example, a schema where a temporary entity is a subclass of a snapshot entity is consistent.

An incorrect ER schema can be the result of disjoint subclasses – i.e., a partitioning. A schema is inconsistent if exactly one of a whole set of snapshot disjoint subclasses is temporary [McBrien *et al.*, 1992]. Without loss of generality, let us illustrate the case where E_1, E_2 are disjoint subclasses of the entity E , with E_1 snapshot and E_2 temporary, then such an ER schema is inconsistent. Indeed, the knowledge base corresponding to this ER schema is unsatisfiable:

$$\begin{aligned} E &\sqsubseteq E_1 \sqcup E_2, & E_1 &\sqsubseteq E \cap \neg E_2, & E_2 &\sqsubseteq E & \text{(disjoint subclass axioms)} \\ E_1 &\sqsubseteq (\Box^+ E_1) \cap (\Box^- E_1) & & & & & \text{(snapshot axiom)} \\ E_2 &\sqsubseteq (\Diamond^+ \neg E_2) \sqcup (\Diamond^- \neg E_2) & & & & & \text{(temporary axiom)} \end{aligned}$$

This can showed by proving that the entity E_2 has no models. Let suppose, by absurd, that a is an instance of E_2 at time t_0 , then, due to the subclass relationship, a is also of kind $E - \{a : E_2, a : E\}_{t_0}$. Since E_2 is a temporary entity there must be t_1 such that $\{a : \neg E_2\}_{t_1}$. As we showed previously, a superclass of a snapshot entity must be a snapshot entity. Then, since E_1 is a snapshot entity, also E must be snapshot, and at t_1 we have $\{a : \neg E_2, a : E\}_{t_1}$. Due to the disjointness subclass axioms, necessarily a is of kind E_1 at $t_1 - \{a : E_1, a : \neg E_2, a : E\}_{t_1}$. But E_1 is a snapshot entity, then a is of kind E_1 at t_0 , too. At the end, at time t_0 we have: $\{a : E_1, a : E_2, a : E\}_{t_0}$, which is a contradiction since E_1, E_2 are disjoint entities. The following is an immediate consequence of the above inconsistent schema:

$$\begin{aligned} \{E &\sqsubseteq E_1 \sqcup E_2, & E_1 &\sqsubseteq E \cap \neg E_2, & E_2 &\sqsubseteq E, \\ E_1 &\sqsubseteq (\Box^+ E_1) \cap (\Box^- E_1)\} & & \models & E_2 &\sqsubseteq (\Box^+ E_2) \cap (\Box^- E_2) \end{aligned}$$

i.e., an ER schema with exactly one entity whose temporal behaviour is unknown among a whole set of snapshot disjoint subclasses, will imply that this entity is necessarily snapshot.

5.2 Snapshot Vs. Temporary Relationships

The case for relationships is more complex. Temporary relationships are captured by enforcing the *temporary axiom* on relationships – in a way analogous to the case of temporary entities:

$$R \sqsubseteq (\Diamond^+ \neg R) \sqcup (\Diamond^- \neg R) \quad (\text{Temporary axiom})$$

To capture snapshot relationships, in addition to the *snapshot axiom*, we need to force each ER-role to be time invariant. For this purpose, the so called *global features* are needed. They are features whose value does not depend on time: we will indicate such particular kind of feature by prefixing the feature name with a “ $\star$ ”. Atomic global features are interpreted as partial functions independent from time: $\forall t, v \in \mathcal{P}. \star g^{\mathcal{I}(t)} = \star g^{\mathcal{I}(v)}$.

Using global features instead of generic features for ER-roles defining a relationship results in a *homogeneous relationship* – i.e., a relationship with tuples whose values are valid at the same time period. Homogeneous relationships are encoded by means of the following axiom:

$$R \sqsubseteq (\star P_{E_1}^R : E_1) \sqcap \dots \sqcap (\star P_{E_n}^R : E_n) \quad (\text{Homogeneity axiom})$$

Snapshot relationships are necessarily global and homogeneous relationships. Thus, if R is a snapshot relationship involving the entities $E_1, \dots, E_n$, the following axioms should be added to Σ :

$$R \sqsubseteq (\Box^+ R) \sqcap (\Box^- R) \quad (\text{Snapshot axiom})$$

$$R \sqsubseteq (\star P_{E_1}^R : E_1) \sqcap \dots \sqcap (\star P_{E_n}^R : E_n) \quad (\text{Homogeneity axiom})$$

The two axioms are such that whenever a tuple belongs to a snapshot relationship, then the very same tuple is assumed to belong to the relationship at every time.

According to the running example, all relationships – with the exception of the **Responsible-for** relationships – have to be modelled as temporary relationships.

The interaction between temporal and snapshot constructs can result in an inconsistent ER schema that can be checked and discarded automatically. This is case when a snapshot relationship R involves a temporary entity. Indeed, the following knowledge base is unsatisfiable:

$$\begin{aligned} R &\sqsubseteq (\Box^+ R) \sqcap (\Box^- R) \\ R &\sqsubseteq (\star P_{E_1}^R : E_1) \sqcap \dots \sqcap (\star P_{E_n}^R : E_n) \\ E_i &\sqsubseteq (\Diamond^+ \neg E_i) \sqcup (\Diamond^- \neg E_i) \end{aligned}$$

i.e., snapshot relationships cannot have temporary entities as participants. This can be formally proved by showing that such a relationship has no models. Let us suppose, by absurd, that a is an instance of R at time t_0 then a is also related to an instance of E_i via the ER-role $\star P_{E_i}^R$, i.e., $\{a : R, a \star P_{E_i}^R b, b : E_i\}_{t_0}$. Since

E#	ENAME	DEPARTMENT	SALARY
121	Tom	$\langle [10, 12], Sales \rangle$	$\langle [10, 15], 20K \rangle$
		$\langle [14, 18], Mktg \rangle$	$\langle [15, 17], 25K \rangle$
			$\langle [17, 20], 30K \rangle$
133	Ann	$\langle [25, 30], Sales \rangle$	$\langle [25, 30], 35K \rangle$
133	John	$\langle [18, now], Toys \rangle$	$\langle [18, now], 42K \rangle$

Fig. 3. The employee heterogeneous relation

E_i is temporary then there must be a time t_1 such that b is not of type $E_i - \{b : \neg E_i\}_{t_1}$. On the other hand, since R is a snapshot relation, a is an instance of R also at time t_1 which is always related via the same instance b to the entity E_i , since the ER roles are global features $- \{a : R, a \star P_{E_i}^R b, b : E_i\}_{t_1}$. This result in a contradiction since b was supposed not to be of type E_i at t_1 . To obtain this inconsistency result the use of global features instead of conventional features is crucial. Indeed, mapping ER-roles of snapshot relationships using conventional features would allow the same R instance to be related to different ER-roles at different times, then invalidating the above expected inference.

On the other hand, temporary relationships admit snapshot entities since the entity instances participate in the relationship only for a temporary time – i.e., during the validity time of the relationship.

6 Heterogeneous Relationships

In the temporal relational database community, a relation is called *heterogeneous* when its attribute values have different period of existence [Tansel and Tin, 1998]. Conversely, *homogeneity* impose that the whole tuple is associated with a unique time stamp. The Figure 3 shows a heterogeneous employee relation as presented in the paper [Tansel and Tin, 1998]. The attributes **DEPARTMENT** and **SALARY** have time stamps represented as intervals with (possible) different values at different time points – as is the case for the tuple associated with *Tom*.

The translation 1 of ER-relationships into *ALCQIT* gives rise to a relationship with (possibly) heterogeneous values. However, relationships can be forced to have *homogeneous* tuples [Gadia, 1988] by using global features to map the ER-roles – as we showed in Section 5.2. Combining both global and generic features we can represent heterogeneous relationships with both time-varying and time-independent attributes. For example, the employee relation in Figure 3, where **E#** and **ENAME** are time-independent attributes, has the following mapping:

$$R \sqsubseteq (\star E\# : ID) \sqcap (\star ENAME : Name) \sqcap (DEPARTMENT : Dept) \sqcap (SALARY : Amount)$$

[McBrien *et al.*, 1992] enforce temporal constraints on the validity time of instances of entities involved in heterogeneous relationships. The original ERT model [Theodoulidis *et al.*, 1991] has been extended to include *historical* marks (H-mark) by imposing particular temporal constraints on the validity time of the different ER-roles. Different temporal constraints give rise to different H-marked relationships. For example, a relationship between entities E_1 and E_2 is H-marked *past* if the instances of E_2 involved in the relationship hold at intervals before the intervals where the instances of E_1 hold. Both historical past and historical future relationships can be formalized with the following axioms:

$$R \sqsubseteq (P_{E_2}^R : E_2 \sqcap \neg P_{E_1}^R : E_1) \sqcap (P_{E_2}^R : E_2 \mathcal{U} P_{E_1}^R : E_1) \text{ (Historical past axiom)}$$

$$R \sqsubseteq (P_{E_1}^R : E_1 \sqcap \neg P_{E_2}^R : E_2) \sqcap (P_{E_1}^R : E_1 \mathcal{U} P_{E_2}^R : E_2) \text{ (Historical future axiom)}$$

i.e., as far as historical past is concerned, the attribute which involve the entity $E_2 - P_{E_2}^R$ – is true *until* the attribute involving $E_1 - P_{E_1}^R$ – holds.

7 Conclusions

This preliminary work gives a logical formalisation of a temporal ER model, which has the ability to express both enhanced temporal ER schemas and (temporal) integrity constraints in the form of general axioms imposed on the schema itself. The formal language we have proposed is a member of the family of Description Logics, and it has a decidable reasoning problem, thus allowing for automated deduction over the whole conceptual representation. We have also shown how the integrity constraints can encode the distinction between time-varying and snapshot constructs.

This work is just at the beginning. The most promising research direction to be explored is to better characterise the expressivity of temporal integrity constraints in order to axiomatise several extensions as proposed in the literature of temporal ER models.

Acknowledgements

The work presented in this paper was partially supported by the “Foundations of Data Warehouse Quality” (DWQ) European ESPRIT IV Long Term Research (LTR) Project 22469.

References

- [Artale and Franconi, 1998] Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research*, 9:463–506, 1998.
- [Artale and Franconi, 1999] Alessandro Artale and Enrico Franconi. Temporal description logics. In L. Vila, P. vanBeek, M. Boddy, M. Fisher, D. Gabbay, A. Galton, and R. Morris, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, 1999. To appear.

- [Calvanese *et al.*, 1994] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class-based representation formalisms. In *Proc. of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, May 1994.
- [Calvanese *et al.*, 1998] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.
- [Calvanese *et al.*, 1999] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, Amsterdam, 1999. To appear.
- [Elmasri and Navathe, 1994] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, 2nd edition, 1994.
- [Gadia, 1988] S.K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions On Database Systems*, 13:418–448, 1988.
- [Gregersen and Jensen, 1999] H Gregersen and J. S. Jensen. Temporal Entity-Relationship models - a survey. *IEEE Transactions on Knowledge and Data Engineering*, 1999. To appear.
- [McBrien *et al.*, 1992] P. McBrien, A.H. Seltveit, and B. Wangler. An Entity-Relationship model extended to describe historical information. In *Proc. of CIS-MOD'92*, pages 244–260, Bangalore, India, 1992.
- [Schild, 1993] Klaus D. Schild. Combining terminological logics with tense logic. In *Proceedings of the 6th Portuguese Conference on Artificial Intelligence, EPIA'93*, October 1993.
- [Tansel and Tin, 1998] A.U. Tansel and E. Tin. Expressive power of temporal relational query languages and temporal completeness. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice*, Lecture Notes in Computer Science, pages 129–149. Springer-Verlag, 1998.
- [Theodoulidis *et al.*, 1991] C. Theodoulidis, P. Loucopoulos, and B. Wangler. A conceptual modelling formalism for temporal database applications. *Information Systems*, 16(3):401–416, 1991.
- [Wolter and Zakharyashev, 1998a] F. Wolter and M. Zakharyashev. Temporalizing description logics. In *Proceedings of FroCoS'98*, Amsterdam, NL, 1998.
- [Wolter and Zakharyashev, 1998b] Frank Wolter and Michael Zakharyashev. Satisfiability problem in description logics with modal operators. In A.Cohn, L.Schubert, and S.C.Shapiro, editors, *Proc. of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 512–523, Trento, Italy, June 1998. Morgan Kaufmann.

Automatic Migration and Wrapping of Database Applications – A Schema Transformation Approach

Peter McBrien¹ and Alexandra Poulovassilis²

¹ Dept. of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ
pjm@doc.ic.ac.uk

² Dept. of Computer Science, Birkbeck College, University of London,
Malet Street, London WC1E 7HX,
ap@dc.s.bbk.ac.uk

Abstract. Integration of heterogeneous databases requires that semantic differences between schemas are resolved by a process of schema transformation. Previously, we have developed a general framework to support the schema transformation process, consisting of a hypergraph-based common data model and a set of primitive schema transformations defined for this model. Higher-level common data models and primitive schema transformations for them can be defined in terms of this lower-level model.

In this paper, we show that a key feature of our framework is that both primitive and composite schema transformations are automatically *reversible*. We show how these transformations can be used to automatically migrate or wrap data, queries and updates between semantically equivalent schemas. We also show how to handle transformations between non-equivalent but overlapping schemas. We describe a prototype schema integration tool that supports this functionality. Finally, we briefly discuss how our approach can be extended to more sophisticated application logic such as constraints, deductive rules, and active rules.

1 Introduction

Common to many methods for integrating heterogeneous data sources is the requirement for **logical integration** [14,6] of the data, due to variations in the design of data models for the same universe of discourse (UoD). Logical integration requires that we are able to transform models to equivalent ones w.r.t. the UoD, and also to translate data, queries or updates on such models. The work in this paper focuses on providing a practical, yet formal, approach to logical integration which directly supports this translation. Our previous work [9,10,12] has provided a general formalism to underpin logical schema transformation and integration. Here we extend this work by providing schema transformations that are automatically reversible. As we will see, this allows the automatic translation of data, queries and updates between equivalent schemas. We also extend our

schema transformations to operate between schemas representing overlapping but non-equivalent UoDs. This means that should two databases vary in their coverage of the UoD we are still able to translate data, queries and updates over their common part of the UoD.

Current implementations of database interoperability, such as InterViso [15], TSIMMIS [2] and Garlic [13,5], are **query-processing oriented**. They focus on providing mechanisms by which a query can be submitted to the federated schema or mediator, and that query be translated to queries on the source databases. In contrast, our approach is **schema-equivalence oriented** in that we focus on providing mechanisms for defining the equivalence between database schemas, and on then using that equivalence to automatically perform whatever transformations on data, queries or updates are necessary. As we will see below, our approach has the further advantage of decomposing the transformation of schemas into a sequence of small steps, whereas the query-processing oriented approaches require that constructs in one schema are directly defined in terms of those in the other schema. We differ from other approaches to ‘automatic wrapper generation’ [16,3] in directly using the basic schema transformation steps to migrate or transform both data, queries and updates — to give what we term application migration and transformation.

The remainder of this paper is as follows. In Section 2 we review the hypergraph data model that underpins our approach and the primitive transformations on schemas defined in this data model. We show how a small extension to our previous framework makes schema transformations reversible. We also extend our previous framework to handle non-equivalent schemas. In Section 3 we show how the reversible schema transformations that result from our framework can be used to automatically migrate or wrap database applications. Section 4 applies our approach to a simple binary ER model, describes a prototype ER schema integration tool, and gives several examples illustrating its use for transforming ER schemas and automatically translating queries, updates and data between them. Section 5 gives our concluding remarks and directions for further work.

2 The Schema Transformation Framework

2.1 Review of Our Previous Work

This section gives a short review of our schema transformation framework, and more details of this material can be found in [12].

A **schema** in the **hypergraph data model** (HDM) is a triple $\langle Nodes, Edges, Constraints \rangle$. A **query** q **over a schema** $S = \langle Nodes, Edges, Constraints \rangle$ is an expression whose variables are members of $Nodes \cup Edges$ ¹. *Nodes* and *Edges* define a labelled, directed, nested hypergraph. It is nested in the sense that edges can link any number of both nodes *and* other edges.

¹ Since what we provide is a framework, the query language is not fixed but will vary between different implementation architectures. In our examples in this paper, we assume that it is the relational calculus.

Constraints is a set of boolean-valued queries over S . Nodes are uniquely identified by their names. Edges and constraints have an optional name associated with them.

An **instance** I of a schema $S = \langle Nodes, Edges, Constraints \rangle$ is a set of sets satisfying the following:

- (i) each construct $c \in Nodes \cup Edges$ has an extent, denoted by $Ext_{S,I}(c)$, that can be derived from I ; ²
- (ii) conversely, each set in I can be derived from the set of extents $\{Ext_{S,I}(c) \mid c \in Nodes \cup Edges\}$;
- (iii) for each $e \in Edge$, $Ext_{S,I}(e)$ contains only values that appear within the extents of the constructs linked by e (domain integrity);
- (iv) the value of every constraint $c \in Constraints$ is true, the **value** of a query q being given by $q[c_1/Ext_{S,I}(c_1), \dots, c_n/Ext_{S,I}(c_n)]$ where $c_1, \dots, c_n$ are the constructs in $Nodes \cup Edges$.

We call the function $Ext_{S,I}$ an **extension mapping**. A **model** is a triple $\langle S, I, Ext_{S,I} \rangle$. Two schemas are **equivalent** if they have the same set of instances. Given a condition f , a schema S **conditionally subsumes** a schema S' w.r.t. f if any instance of S' satisfying f is also an instance of S . Two schemas S and S' are **conditionally equivalent** w.r.t. f if they each conditionally subsume each other w.r.t. f .

We first developed these definitions of schemas, instances, and schema equivalence in the context of an ER common data model, in earlier work [9,10]. A comparison with other approaches to schema equivalence and schema transformation can be found in [10], which also discusses how our framework can be applied to schema integration.

We now list the primitive transformations of the HDM. Each transformation is a function that when applied to a model returns a new model. Each transformation has a proviso associated with it which states when the transformation is **successful**. Unsuccessful transformations return an “undefined” model, denoted by ϕ . Any transformation applied to ϕ returns ϕ .

1. *renameNode* $\langle fromName, toName \rangle$ renames a node. Proviso: *toName* is not already the name of some node.
2. *renameEdge* $\langle \langle fromName, c_1, \dots, c_m \rangle, toName \rangle$ renames an edge. Proviso: *toName* is not already the name of some edge.
3. *addConstraint* c adds a new constraint c . Proviso: c evaluates to true.
4. *delConstraint* c deletes a constraint. Proviso: c exists.
5. *addNode* $\langle name, q \rangle$ adds a node named *name* whose extent is given by the value of the query q . Proviso: a node of that name does not already exist.
6. *delNode* $\langle name, q \rangle$ deletes a node. Here, q is a query that states how the extent of the deleted node could be recovered from the extents of the remaining schema constructs (thus, not violating property (ii) of an instance). Proviso: the node exists and participates in no edges.

² Again, the language in which this derivation mapping is defined, and also that in point (ii), is not fixed by our framework.

7. *addEdge* $\langle \langle name, c_1, \dots, c_m \rangle, q \rangle$ adds a new edge between a sequence of existing schema constructs $c_1, \dots, c_m$. The extent of the edge is given by the value of the query q . Proviso: the edge does not already exist, $c_1, \dots, c_m$ exist, and q satisfies the appropriate domain constraints.
8. *delEdge* $\langle \langle name, c_1, \dots, c_m \rangle, q \rangle$ deletes an edge. q states how the extent of the deleted edge could be recovered from the extents of the remaining schema constructs. Proviso: the edge exists and participates in no edges.

For each of these transformations, there is a 3-ary version which takes as an extra argument a condition which must be satisfied in order for the transformation to be successful.

A **composite transformation** is a sequence of $n \geq 1$ primitive transformations. A transformation t is **schema-dependent (s-d)** w.r.t. a schema S if t does not return ϕ for any model of S , otherwise t is **instance-dependent (i-d)** w.r.t. S . It is easy to see that if a schema S can be transformed to a schema S' by means of a s-d transformation, and vice versa, then S and S' are equivalent. If S can be transformed to S' by means of an i-d transformation with proviso f , and vice versa, then S and S' are conditionally equivalent w.r.t f .

Example 1 Transforming between two schemas Consider two schemas

$S_1 = \langle N_1, E_1, C_1 \rangle$ and $S_2 = \langle N_2, E_2, C_2 \rangle$ where

$N_1 = \{person, mathematician, compScientist\}$, $E_1 = \{\}$,

$C_1 = \{mathematician \subseteq person, compScientist \subseteq person\}$,

$N_2 = \{person, dept\}$, $E_2 = \{\langle Null, person, dept \rangle\}$, $C_2 = \{\}$.

The transformation T_{S_1, S_2} below transforms S_1 to S_2 and is s-d:

$T_{S_1, S_2} =$
addNode $\langle dept, \{maths, compsci\} \rangle;$
addEdge $\langle Null, person, dept, \{ \langle x, maths \rangle \mid x \in mathematician \} \cup \{ \langle x, compsci \rangle \mid x \in compScientist \} \rangle;$
delConstraint $(compScientist \subseteq person);$
delConstraint $(mathematician \subseteq person);$
delNode $\langle compScientist, \{x \mid \langle x, compsci \rangle \in \langle Null, person, dept \rangle\} \rangle$
delNode $\langle mathematician, \{x \mid \langle x, maths \rangle \in \langle Null, person, dept \rangle\} \rangle;$

Conversely, the transformation T_{S_2, S_1} below transforms S_2 to S_1 and is i-d since there is a condition $dept = \{maths, compsci\}$ on the last-but-one step:

$T_{S_2, S_1} =$
addNode $\langle mathematician, \{x \mid \langle x, maths \rangle \in \langle Null, person, dept \rangle\} \rangle;$
addNode $\langle compScientist, \{x \mid \langle x, compsci \rangle \in \langle Null, person, dept \rangle\} \rangle;$
addConstraint $(mathematician \subseteq person);$
addConstraint $(compScientist \subseteq person);$
delEdge $\langle Null, person, dept, \{ \langle x, maths \rangle \mid x \in mathematician \} \cup \{ \langle x, compsci \rangle \mid x \in compScientist \} \rangle$ ($dept = \{maths, compsci\}$);
delNode $\langle dept, \{maths, compsci\} \rangle$

Thus S_1 and S_2 are c-equivalent.

2.2 Reversibility of Schema Transformations

There are two minor but significant differences between the transformations listed in the previous section and those given in [12]. Firstly, the *delNode* and *delEdge* transformations now require a restoring query for the deleted node or edge to be specified. Secondly, addition and deletion transformations are only successful if the construct being added (deleted) does not exist (exists) in the schema, and similarly renamings of constructs must be to new names. These changes allow the reverse transformation to the original model to be automatically derivable. In particular, for every primitive transformation t such that $t(\langle S, I, Ext_{S,I} \rangle) \neq \phi$ there exists a **reverse primitive transformation** t^{-1} such that $t^{-1}(t(\langle S, I, Ext_{S,I} \rangle)) = \langle S, I, Ext_{S,I} \rangle$. We list the reverse transformation of each primitive transformation below. Notice that if t depends on a condition c , since t is successful c must necessarily hold and so need not be verified within t^{-1} :

Transformation (t)	Reverse Transformation (t^{-1})
<i>renameNode</i> $\langle from, to \rangle c$	<i>renameNode</i> $\langle to, from \rangle$
<i>renameEdge</i> $\langle \langle from, schemes \rangle, to \rangle c$	<i>renameEdge</i> $\langle \langle to, schemes \rangle, from \rangle$
<i>addConstraint</i> $q c$	<i>delConstraint</i> q
<i>delConstraint</i> $q c$	<i>addConstraint</i> q
<i>addNode</i> $\langle n, q \rangle c$	<i>delNode</i> $\langle n, q \rangle$
<i>delNode</i> $\langle n, q \rangle c$	<i>addNode</i> $\langle n, q \rangle$
<i>addEdge</i> $\langle e, q \rangle c$	<i>delEdge</i> $\langle e, q \rangle$
<i>delEdge</i> $\langle e, q \rangle c$	<i>addEdge</i> $\langle e, q \rangle$

The reversibility of primitive transformations generalises to any successful composite transformation: given any such composite transformation $t_1; \dots; t_n$ its reverse composite transformation is $t_n^{-1}; \dots; t_1^{-1}$. Thus, in Example 1, $T_{S_2, S_1}^{-1} = T_{S_1, S_2}$ and T_{S_1, S_2}^{-1} is T_{S_2, S_1} but without the condition $dept = \{maths, compsci\}$ in the last-but-one step.

2.3 Transforming between Non-equivalent Schemas

Even when two databases are designed to hold the same information, it is likely that they will differ slightly and will not be precisely equivalent. We define four more low-level transformations to deal with such situations. If a schema S_2 contains a construct (*i.e.* node or edge in the low-level framework) which cannot be derived from a schema S_1 , we say that S_2 is formed by **extending** S_1 with that construct, giving the value **void** as the query that creates the extent of the new construct. The reverse transformation involves **contracting** S_2 to obtain S_1 , and the query is again **void** indicating that the deleted construct cannot be restored from S_1 . The four new low-level transformations can be defined in terms of the existing ones as follows:

$$\begin{aligned}
 extendNode\ n &= addNode\ \langle n, void \rangle & extendEdge\ e &= addEdge\ \langle e, void \rangle \\
 contractNode\ n &= delNode\ \langle n, void \rangle & contractEdge\ e &= delEdge\ \langle e, void \rangle
 \end{aligned}$$

If a compound transformation t such that $S_2 = t(S_1)$ contains a transformation of the form *contractNode* n or *contractEdge* e then S_1 can be regarded as

extending S_2 with respect to n or e , in the sense that S_1 holds information on n or e that cannot be held in S_2 .

Similarly, if a compound transformation t such that $S_2 = t(S_1)$ contains a transformations of the form *extendNode* n or *extendEdge* e then S_2 can be regarded as **extending S_1 with respect to n or e** , in the sense that S_2 holds information in n or e that cannot be held in S_1 .

Clearly both of the above may apply for any compound transformation t , giving rise to a pair of schemas which are **overlapping**. For such pairs of schemas there will be some queries that can only be answered by performing a join between the two schemas.

Suppose that pre-integration transformations have been applied to a set of source schemas so that all conflicts between them (both structural and constraint-based) have been removed and the schemas can be integrated by forming a union of their *Nodes*, *Edges* and *Constraints* components. Then the resulting **federated schema**, S_f , has the properties that (i) there is no node or edge with respect to which S_f extends all of the source schemas (**minimality**), and (ii) there is no node or edge with respect to which a source schema extends S_f (**completeness**). In practice, property (ii) is often not implemented when a partial federated schema is constructed for some specific application.

3 Migrating and Wrapping Databases

If a schema S_1 is transformed into an equivalent one S_2 , we may want to migrate (*i.e.* re-engineer) an existing database application from S_1 to S_2 . The old schema and extension become obsolete and the new schema and new extension become the operational database. In this case, we will see below how the transformation from S_1 to S_2 can be used to automatically:

- (i) migrate the database extension associated with S_1 to populate a new database extension associated with S_2 ;
- (ii) migrate queries posed on S_1 to queries posed on S_2 ;
- (iii) migrate updates posed on S_1 to updates posed on S_2 .

An alternative scenario is that an existing database is being “wrapped” by a new, equivalent schema. The old schema and extension carry on being operational but clients are able to interact with them via the new schema. In this case, we will see below how the transformation from S_1 to S_2 can be used to automatically:

- (iv) translate queries posed on S_2 to queries posed on S_1 ;
- (v) translate updates posed on S_2 to updates posed on S_1 .

3.1 Data Migration

Suppose a schema S_1 is transformed into an equivalent one S_2 and we wish to automatically migrate the current extension of S_1 to S_2 . Consider first the case that S_1 is transformed to S_2 by a single primitive transformation. The only cases we need to consider are the addition of a new node or edge:

- *addNode* $\langle n, q \rangle$: the new node n is populated by evaluating q on the extension of S_1 .
- *addEdge* $\langle e, q \rangle$: the new edge e is similarly populated by evaluating q on the extension of S_1 .

For composite transformations, a new extent is generated each time an *addNode* or *addEdge* transformation is applied.

For example, if S_1 is transformed to S_2 by means of T_{S_1, S_2} in Example 1 then the queries specified in the first and second steps of T_{S_1, S_2} can be used to automatically derive $Ext_{S_2, I}$:

$$Ext_{S_2, I}(dept) = \{maths, compsci\}$$

$$Ext_{S_2, I}(\langle Null, person, dept \rangle) = \{\langle x, maths \rangle \mid x \in Ext_{S_1, I}(mathematician)\} \cup \{\langle x, compsci \rangle \mid x \in Ext_{S_1, I}(compScientist)\}$$

Notice that $Ext_{S_2, I}$ is precisely as defined in Example 1, except that it has now been automatically derived from the schema transformation steps.

3.2 Query and Update Migration

Suppose a schema S_1 is transformed into an equivalent one S_2 and the extension of S_1 has been migrated to S_2 . We can also automatically migrate queries posed on S_1 to ones posed on S_2 .

Consider first the case that S_1 is transformed to S_2 by a single primitive transformation. The only cases we need to consider in order to migrate a query q_1 posed on S_1 to an equivalent query q_2 posed on S_2 are to apply renamings and to substitute occurrences of a deleted node or edge by their restoring query:

- *renameNode* $\langle from, to \rangle$: $q_2 = [from/to]q_1$.
- *renameEdge* $\langle \langle from, schemes \rangle, to \rangle$: $q_2 = [\langle from, schemes \rangle / \langle to, schemes \rangle]q_1$.
- *delNode* $\langle n, q \rangle$: $q_2 = [n/q]q_1$.
- *delEdge* $\langle e, q \rangle$: $q_2 = [e/q]q_1$.

For composite transformations, the above substitutions are successively applied in order to obtain the final migrated query q_2 .

Consider now an update u_1 posed on S_1 , taking one of the following general forms, where q is a query, n a node of S_1 and e an edge of S_1 :

$$insert\ q\ n, \ insert\ q\ e, \ delete\ q\ n, \ delete\ q\ e$$

Then exactly the same substitutions as for queries above can be applied to u_1 in order to obtain an equivalent update u_2 posed on S_2 . Of course u_2 will be unambiguous only if S_1 is an updateable view of S_2 , otherwise all the usual problems associated with view updates [4,7] will need to be addressed.

To illustrate, the following query on S_1 returns people who work in either the Computer Science and or the Maths department:

$$\text{compScientist} \cup \text{mathematician}$$

The transformation T_{S_1, S_2} results in the following substitution:

$$\begin{aligned} &[\text{mathematician} / \{x \mid \langle x, \text{maths} \rangle \in \langle \text{Null}, \text{person}, \text{dept} \rangle\}, \\ &\text{compScientist} / \{x \mid \langle x, \text{compsci} \rangle \in \langle \text{Null}, \text{person}, \text{dept} \rangle\}] \end{aligned}$$

which when applied to the query results in the following equivalent query on S_2 :

$$\begin{aligned} &\{x \mid \langle x, \text{compsci} \rangle \in \langle \text{Null}, \text{person}, \text{dept} \rangle\} \cup \{x \mid \langle x, \text{maths} \rangle \\ &\quad \in \langle \text{Null}, \text{person}, \text{dept} \rangle\} \end{aligned}$$

Similarly, this update on S_1 adds *Bob* to the extent of *mathematician*:

$$\text{insert Bob mathematician}$$

Applying the above substitution results in the following update on S_2 :

$$\text{insert Bob } \{x \mid \langle x, \text{maths} \rangle \in \langle \text{Null}, \text{person}, \text{dept} \rangle\}$$

and this update has the unambiguous meaning that the tuple $\langle \text{Bob}, \text{maths} \rangle$ should be added to the extent of $\langle \text{Null}, \text{person}, \text{dept} \rangle$.

3.3 Query and Update Translation

Suppose now a schema S_1 is transformed into an equivalent one S_2 but that the extension of S_1 is not migrated to S_2 . Then we can automatically translate queries and updates posed on S_2 to ones posed on S_1 , as follows.

We first automatically derive the reverse transformation from S_2 to S_1 , as shown in Section 2.2. In order to translate queries posed on S_2 to queries posed on S_1 we then use exactly the same method as for migrating queries from S_1 to S_2 in Section 3.2 above, except that now it is with reference to the reverse transformation. Updates posed on S_2 are similarly be translated to updates posed on S_1 and will be unambiguous if S_2 is an updateable view of S_1 .

4 A Prototype ER Schema Transformation Tool

In [12] we show how to use the HDM to define transformations in a rich ER modelling language supporting n-ary relations, attributes on relations, generalisation, and complex attributes. That paper, together with [10], show the semantic richness of our framework by defining, and thereby formalising, the main ER schema equivalences that have appeared in literature. We do not repeat this work here and instead focus on the reversibility of ER schema transformations, and the fact that they can be used to migrate or wrap databases whose schemas

are defined using an ER data model. We consider a simple ER modelling language supporting binary relationships and generalisation hierarchies. The primitive transformations for this language are: $rename_E$, $rename_A$, $rename_R$, for renaming an entity class, attribute and relationship, respectively; add_E , add_A , add_R , add_G for adding an entity class, attribute, relationship or generalisation hierarchy; and $del_E/A/R/G$ for deleting such constructs. The definitions of these transformations in terms of the lower-level HDM primitive transformations can be found in [12].

The notions of instances, models, schema equivalence, and s-d/i-d transformations extend naturally to ER schemas. Thus, if an ER schema S can be transformed to an ER schema S' by means of an s-d transformation, and vice versa, then S and S' are u-equivalent. Similarly, if S can be transformed to S' by means of an i-d transformation with proviso f , and vice versa, then S and S' are c-equivalent w.r.t f .

Migration and wrapping are handled in the same way at this higher semantic level as in the low-level framework. For data migration, new constructs created by add_E , add_A or add_R transformations are populated by evaluating the supplied query (add_G transformations simply add a number of constraints to the schema and do not have an associated extent). For query and update migration from the original to the resulting schema, we apply renamings and substitute occurrences of a deleted entity, attribute or relationship by their restoring query. For query and update translation from the resulting to the original schema, we use the same technique but with reference to the reverse transformation. Several examples of migrating and wrapping at this higher semantic level are given in the next section, where we describe a prototype schema integration tool.

4.1 A Prototype ER Schema Integration Tool

We have implemented a prototype ER schema integration tool that supports the functionality described above, with some minor departures. The tool provides the database integrator with the following set of primitive ER schema transformations: ³

```
addEntity(Ss, Sn, Scheme, CSs, CSn, Query)
delEntity(Sn, Ss, Scheme, CSn, CSs, Query)
addRelationship(Ss, Sn, Scheme, Card, CSs, CSn, Query)
delRelationship(Sn, Ss, Scheme, Card, CSn, CSs, Query)
addAttribute(Ss, Sn, Scheme, Card, CSs, CSn, Query)
delAttribute(Sn, Ss, Scheme, Card, CSn, CSs, Query)
addGeneralisation(Ss, Sn, Scheme, CSn)
delGeneralisation(Sn, Ss, CSn, Scheme)
```

The tool also provides a set of renaming transformations, but we will not be using these for our examples here and refer the reader to [8] for details of

³ The tool is implemented in Edinburgh Prolog, in which identifiers beginning with an upper case letter are variables, identifiers beginning with a lower case letter or enclosed in single quotes are constants, and the underscore character is an anonymous variable.

them. The parameters of the above primitive transformations have the following meaning:

Scheme identifies the **subject construct**. This is the construct being added or deleted by the transformation. **Scheme** includes variable(s) used to instantiate instances of the construct and it takes one of the following forms:

- `[ent, Name, E]` represents an entity type called `Name`, and `E` may be instantiated with instances of `Name`.
- `[att, EName, AName, E, A]` represents an attribute `AName` of entity type `EName`, where `A` may be instantiated with a value of the attribute associated with the instance `E` of `EName`.
- `[rel, E1Name, RName, E2Name, E1, E2]` represents a relationship `RName` between entities `E1Name` and `E2Name`. `E1` and `E2` may be instantiated with pairs of entity instances involved in the relationship.
- `[gen, GenType, Super, SubList]` represents a generalisation between a super-entity type `Super` and the list of sub-entity types `SubList`. `GenType` may be either **total** or **partial**.

When anonymous free variables appear in a **Scheme** e.g. `[att, person, name, _, _]`, we may abbreviate the scheme to exclude those variables e.g. `[att, person, name]`. Also, when a scheme is passed to a transformation its first component may be omitted as it can be inferred e.g. we need only write `[person, E]` instead of `[ent, person, E]`.

Ss is what we term the **subject schema** of the transformation and **Sn** is the **non-subject schema** of the transformation. For addition transformations **Sn** is the input schema of the transformation (the one not containing the subject construct) and **Ss** is the output schema. Conversely, for deletion transformations **Ss** is the input schema (the one containing the subject construct) and **Sn** is the output schema.

Card is a pair of cardinality constraints `[C1, C2]`, where `C1` and `C2` are each one of `[1, 1]`, `[1, 1]`, `[1, n]`, or `[1, n]`, with their usual meaning. Hence a scheme `[rel, person, worksin, dept, P, D]` and cardinality constraints `[[1, 1], [1, n]]` indicates that each **person** instance `P` is associated with exactly one **dept** instance `D` and that each **dept** instance is associated with one or more **person** instances.

CSn and **CSs** give constraints, placed on **Sn** and **Ss** respectively, which must be satisfied for the transformation to be successful. The tool here extends the theoretical treatment given in previous sections by allowing the user to provide constraints on *both* the input and the output schema, as opposed to just the former. This makes the transformations between schemas bi-directional, as opposed to uni-directional, and there is really no distinction between the input and the output schemas.

Query is a query expression over the non-subject schema which defines the extent of the subject construct. The syntax of a **Query** is:

```
Query ::= Scheme | Predicate | [or, Query, Query] |
        [and, Query, Query] | [implies, Query, Query] | [not, Query]
```

`Predicate ::= [eq,Atom,Atom] | [less,Atom,Atom]`

4.2 Transformations between Equivalent Schemas

A sequence of transformations made between two schemas will draw an equivalence between those schemas when none of the steps involves a contract or extend transformation. We illustrate in Example 2 one such sequence for the pair of schemas *s1* and *s2* illustrated in Figure 1. We will later show in Section 4.5 how we may transform these schemas to a non-equivalent *s3*.

Example 2 Transforming *s1* to *s2*

- ① Add a new entity type *male* to schema *s1*, to construct a new schema *s1a*. The instances of the new entity type (the variable *EV*) are obtained by taking those instances of *person* which have value *m* for their *sex* attribute. There is no condition on the transformation (represented by []). In the forward direction (*i.e.* transforming *s1* to *s1a*), any value other than *m* for the *sex* attribute will mean that a person will not be classified as *male*. In the reverse direction (*i.e.* *s1a* to *s1*), any instance of *male* will result in the *sex* attribute of that person taking the value *m*.
`addEntity(s1a,s1,[male,EV],[],[],[att,person,sex,EV,m]),`
- ② The *female* entity type is added similarly to *s1a* to construct a new schema *s1b*:
`addEntity(s1b,s1a,[female,EV],[],[],[att,person,sex,EV,f]),`
- ③ Add that *person* is a generalisation of *male* and *female*:
`addGeneralisation(s1c,s1b,[total,person,[male,female]],`
`[implies,[att,person,sex,_,AV],[or,[eq,AV,m],[eq,AV,f]])],`
- ④ Remove the now redundant *sex* attribute from *person*:
`delAttribute(s4,s1c,[person,sex,EV,AV],[[1,1],[1,n]],[],[`
`[or,[and,[ent,male,EV],[eq,AV,m]],`
`[and,[ent,female,EV],[eq,AV,f]])],`

At this stage we have transformed *s1* into the intermediate schema *s4* shown in Figure 1. Notice that the four steps ①- ④ implement the well-known equivalence between attributes and generalisation hierarchies [1].

- ⑤ Add the attribute *dname* to *person*:
`addAttribute(s2c,s4,[person,dname,X,Y],[[1,1],[1,n]],[],[`
`[rel,person,worksin,dept,X,Y]),`
- ⑥ Delete the now redundant *worksin* relationship:
`delRelationship(s2b,s2c,[person,worksin,dept,X,Y],`
`[1,1],[1,n],[],[],[att,person,dname,X,Y]),`
- ⑦ Delete the redundant *dname* attribute on *dept*:
`delAttribute(s2a,s2b,[dept,dname,X,Y],[[1,1],[1,n]],`
`[implies,[att,dept,dname,X,Y],[eq,X,Y]],[`
`[att,person,dname,X,Y]),`
- ⑧ Delete *dept*:

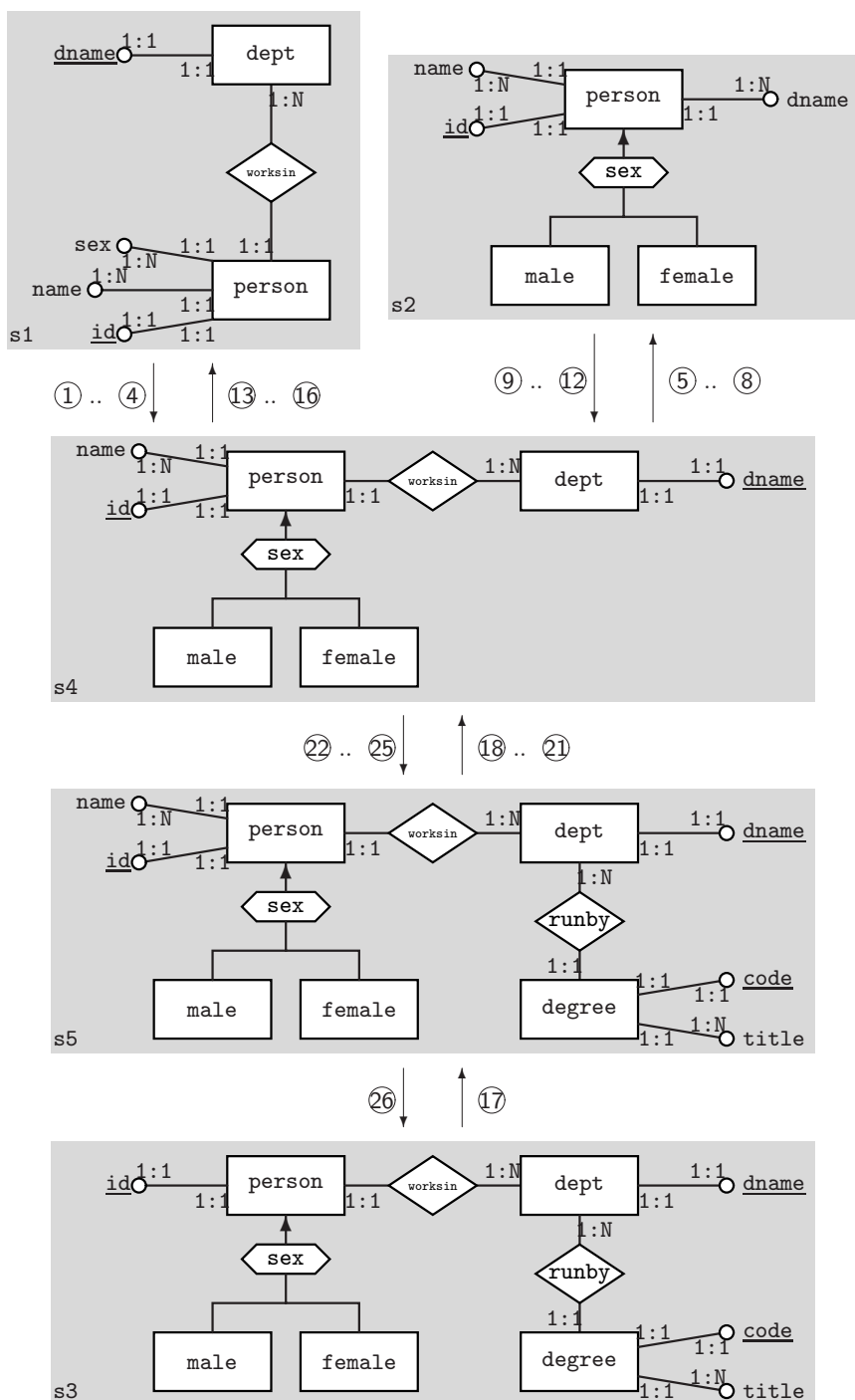


Fig. 1. Example ER schemas, and the transformations that relate them

```
delEntity(s2,s2a,[dept,Y],[],[],[att,person,dname,_,Y]).
```

The four steps ⑤- ⑧ taken as a compound transformation implement the well-known equivalence between attributes and entity classes [1].

Example 3 below gives the reverse transformation of Example 2 which is automatically derived by the tool. Conversely, the database integrator could have specified Example 3 and the tool would then automatically derive Example 2.

Example 3 Transforming s2 to s1

- ⑨ addEntity(s2a,s2,[dept,Y],[],[],[att,person,dname,_,Y]),
- ⑩ addAttribute(s2b,s2a,[dept,dname,X,Y],[[1,1],[1,1]],
[],[implies,[att,dept,dname,X,Y],[eq,X,Y]],
[att,person,dname,X,Y]),
- ⑪ addRelationship(s2c,s2b,[person,worksin,dept,X,Y],
[[1,1],[1,n]],[],[att,person,dname,X,Y]),
- ⑫ delAttribute(s4,s2c,[person,dname,X,Y],[[1,1],[1,n]],[],[rel,
person,worksin,dept,X,Y]),
- ⑬ addAttribute(s1c,s4,[person,sex,EV,AV],[[1,1],[1,n]],[],[or,
[and,[ent,male,EV],[eq,AV,m]],
[and,[ent,female,EV],[eq,AV,f]]]),
- ⑭ delGeneralisation(s1b,s1c,[total,person,[male,female]],
[implies,[att,person,sex,_,AV],[or,[eq,AV,m],[eq,AV,f]]]),
- ⑮ delEntity(s1a,s1b,[female,EV],[],[att,person,sex,EV,f]),
- ⑯ delEntity(s1,s1a,[male,EV],[],[att,person,sex,EV,m]).

4.3 Data Migration

Example 4 below shows how the extent of each construct in s2 is defined in terms of the extents of constructs in s1, together with which step (if any) of the transformation from s1 to s2 is used. Example 5 similarly shows how the extent of each construct in s1 is defined in terms of the extents of constructs in s2, together with which step of the transformation from s2 to s1 is used.

Example 4 Migrating data from s1 to s2

s2	s1	Step
[ent,person,X]	[ent,person,X]	-
[att,person,id,X,Y]	[att,person,id,X,Y]	-
[att,person,name,X,Y]	[att,person,name,X,Y]	-
[att,person,dname,X,Y]	[rel,person,worksin,dept,X,Y]	⑤
[ent,male,X]	[att,person,sex,EV,m]	①
[ent,female,X]	[att,person,sex,EV,f]	②

Example 5 Migrating data from s2 to s1

s1	s2	Step
[ent, person, X]	[ent, person, X]	-
[att, person, id, X, Y]	[att, person, id, X, Y]	-
[att, person, name, X, Y]	[att, person, name, X, Y]	-
[att, person, sex, X, Y]	[or, [and, [ent, male, X], [eq, Y, m]], [and, [ent, female, X], [eq, Y, f]]]	⑬
[rel, person, worksin, dept]	[att, person, dname, X, Y]	⑪
[ent, dept, Y]	[att, person, dname, -, Y]	⑨
[att, dept, dname, X, Y]	[att, person, dname, X, Y]	⑩

4.4 Query and Update Translation

Since the tool handles bi-directional transformations, there is no distinction between migration and translation of queries and updates, so we use the latter term. For any query on **s1**, the table in Example 5 can be used to translate constructs of **s1** into ones of **s2**, resulting in a query on **s2**. Conversely, for any query on **s2**, the table in Example 4 can be used to translate constructs of **s2** into ones of **s1**, resulting in a query on **s1**.

Example 6 Translation of queries from s1 to s2 “Find the ids of all females that work in Computing.” (Since *id* is the key attribute of *person*, we assume that person entities are identical to their *id* attributes.)

Query on s1	Step	Query on s2
[and,	-	[and,
[att, person, sex, Id, f],	⑬	[or, [and, [ent, male, Id], [eq, f, m]],
		[and, [ent, female, Id], [eq, f, f]]],
[rel, person, worksin,	⑪	[att, person, dname, Id, maths]]
dept, Id, computing]]		

The same method as for query translation can be used to translate updates:

Example 7 Translation of inserts from s2 to s1 An insertion on **s2** of a male person with *id* 1000 and name ‘Peter’, can be translated into an insertion on **s1** of a person with with *id* 1000, name ‘Peter’ and *sex* m:

Insert into s2	Step	Insert into s1
[and,	-	[and,
[ent, person, 1000],	-	[ent, person, 1000],
[and,	-	[and,
[ent, male, 1000],	①	[att, person, sex, 1000, m],
[att, person, name,	-	[att, person, name,
1000, ‘Peter’]]]	-	1000, ‘Peter’]]]

4.5 Transformations between Non-equivalent Schemas

Higher-level transformations such as *extendEntity*, *contractEntity*, *extendRelationship*, *contractRelationship*, and so forth are straightforwardly defined in terms

of the low-level *extend* and *contract* ones. To illustrate their use, Example 8 below shows how we may transform schema *s3* in Figure 1 to schema *s4*.

Example 8 Transforming *s3* to *s4*

- ⑪ `expandAttribute(s5,s3,[person,name,-,-])`.
- ⑫ `contractAttribute(s5a,s5,[degree,code,-,-])`,
- ⑬ `contractAttribute(s5b,s5a,[degree,title,-,-])`,
- ⑭ `contractRelationship(s3c,s3b,[degree,runby,dept,-,-])`,
- ⑮ `contractEntity(s4,s5c,[degree,-,-])`,

s4 is to one of the intermediate schemas produced in Examples 2 and 3. Thus, we can use steps 17-21 above followed by steps 13-16 in Example 3 to transform *s3* to *s1* and steps 17-21 followed by steps 5-8 in Example 2 to transform *s3* to *s2*. As before, the reverse steps 22-26 below may be derived from 17-21, to transform *s4* to *s3*:

Example 9 Transforming *s4* to *s3*

- ⑯ `expandEntity(s4,s5c,[degree,-,-])`,
- ⑰ `expandRelationship(s5c,s3b,[degree,runby,dept,-,-])`,
- ⑱ `expandAttribute(s5b,s5a,[degree,title,-,-])`,
- ⑲ `expandAttribute(s5a,s5,[degree,code,-,-])`,
- ⑳ `contractAttribute(s5,s3,[person,name,-,-])`.

Data, updates and queries can be translated between pairs of non-equivalent schemas so long as this is confined to their common constructs. If a query uses some construct which has no derivation in the target schema, as for example in Example 10 below, then the resulting query will contain a *void* term, which our tool will return as invalid (in a production version of the tool it may be possible to give partial answers, using those parts of the query that were successfully translated).

Example 10 Invalid translation of queries from *s3* to *s1* “Find the *id*’s of the persons and the *dname* of departments which are all involved with the degree programme with *code*=‘G500’.”

Query on *s3*

```
[and,
  [rel,person,
    worksin,dept,Id,DName],
  [rel,degree,runby,dept,'G500',F]]
```

Step Query on *s1*

```
- [and,
-   [rel,person,
      worksin,dept,Id,DName],
  ⑳ [void]]
```

4.6 Global Schemas and Queries

In Figure 1, schema *s5* may be regarded as a federated schema which contains the union of the information of the three source schemas *s1*, *s2* and *s3*. Since *s5* extends (as defined in Section 2.3) all three source schemas, there will be some

global queries on **s5** which will result in **void** queries whichever source schema they are mapped onto.

To illustrate, Example 11 below first shows how a global query on **s5** would be translated to each of the three source schemas. It then shows how these three individual translations can be amalgamated into an overall global query plan. The construct **[ask,schema,query]** identifies that a sub-query can be directed to a particular database. If more than one database contains the information, the various alternative queries are placed in construct **[plan,ask1,...,askn]**, indicating that some further query planning is required in order to choose one of the queries for execution. At this stage, consideration of the query processing capabilities of the various databases needs to be taken into account.

Example 11 Query decomposition “Find the **names** and **ids** of the persons and the **dname** of departments which are all involved with the degree programme with **code=‘G500’**.”

Query on s5	Step Query on s1
[and,	- [and,
[att,person,name,Id,Name],	- [att,person,name,Id,Name],
[and,	- [and,
[rel,person,worksin,	- [rel,person,worksin,
dept,Id,DName],	dept,Id,DName],
[rel,degree,runby,	②⑥ [void]
dept,'G500',DName]]]	]]
Query on s5	Step Query on s2
[and,	- [and,
[att,person,name,Id,Name],	- [att,person,name,Id,Name],
[and,	- [and,
[rel,person,worksin	①① [att,person,
dept,Id,DName],	dname,Id,DName],
[rel,degree,runby	②⑥ [void]
dept,'G500',DName]]]	]]
Query on s5	Step Query on s3
[and,	- [and,
[att,person,name,Id,Name],	②⑥ [void],
[and,	- [and,
[rel,person,	- [rel,person,
worksin,dept,Id,DName],	worksin,dept,Id,DName],
[rel,degree,runby,	- [rel,degree,runby,
dept,'G500',DName]]]	- dept,'G500',DName]]]

Query on s5	Global Query Plan
[and,	[and,
[att, person, name,	[plan,
Id, Name],	[ask, s1, [att, person, name, Id, Name]],
	[ask, s2, [att, person, name, Id, Name]]
	],
[and,	[and,
[rel, person, worksin,	[plan,
dept, Id, DName],	[ask, s1,
	[rel, person, worksin, dept, Id, DName]],
	[ask, s2,
	[att, person, dname, Id, DName]],
	[ask, s3,
	[rel, person, worksin, dept, Id, DName]]
	],
[rel, degree, runby,	[ask, s3, [rel, degree, runby,
dept, 'G500', DName]	dept, 'G500', DName]]
]]	]]

5 Conclusions

This paper has described a formal framework for schema transformation which allows the reverse of a schema transformation to be derived automatically. This reversibility allows data, queries and updates to be automatically migrated or translated in either direction between two equivalent schemas S_1 and S_2 provided that a transformation between them has been defined. We have demonstrated how the approach can also be used when S_1 and S_2 are not equivalent, and we have illustrated the approach in use in a simple prototype schema integration tool. The approach can readily be extended to translate more sophisticated application logic such as constraints, deductive rules and active rules.

Our approach is readily applicable within all the main database interoperation architectures [6]. For example, in mediator architectures, it can be used to automatically generate those parts of wrappers that handle the translation between different semantic models. It can also be used to help handle changes in source databases: if a change of a database schema S to a schema S' is represented by a transformation from S to S' then any query that previously was used on S can be automatically translated to a query on S' .

In more recent work [11], we have developed a generic framework for defining the semantics of higher-level modelling language constructs in terms of the HDM. The definitions of the higher-level primitive transformations are then automatically derived. In that paper we also show how these transformations can be used to map between schemas expressed in different modelling languages. In combination with the work described here, this allows us to automatically transform queries between schemas defined in different modelling languages. Also, it is possible to define inter-model links, which support the development of stronger

coupling between different modelling languages than is provided by current approaches.

Our future work will follow two main directions:

- *Embedding the Primitive Transformations into a Programming Language*: The primitive transformations can be extended into a full language for writing schema transformation programs e.g. by adding an iteration construct, a conditional branching construct, and procedures. This would allow the definition of more flexible general-purpose transformations.
- *A Graphical Tool*: The prototype tool we have described here can be used as the basis for a more sophisticated tool, for example one supporting the graphical display and manipulation of schemas, and predefined templates for the common schema transformations.

References

1. C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986. 106, 108
2. S.S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J.D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, October 1994. 97
3. T. Critchlow, M. Ganesh, and R. Musick. Automatic generation of warehouse mediators using an ontology engine. In *Proceedings of the 5th International Workshop on Knowledge Representation Meets Databases (KRDB '98)*, volume 10. CEUR Workshop Proceedings, 1998. 97
4. U. Dayal and P. Bernstein. On the updatability of relational views. In *Proc. 5th International Conference on Very Large Data Bases (VLDB '79)*, Rio de Janeiro, Brazil, October 1979. 102
5. L.M. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proceedings of the 23rd VLDB Conference*, pages 276–285, Athens, Greece, 1997. 97
6. R. Hull. Managing sematic heterogeneity in databases: A theoretical perspective. In *Proceedings of PODS*, 1997. 96, 112
7. R. Langerak. View updates in relational databases with independent scheme. *ACM Transactions on Database Systems*, 15(1), March 1990. 102
8. P.J. McBrien. A reference implementation for a formal approach to schema transformation. Technical Report TR98-09, King's College London, 1998. 104
9. P.J. McBrien and A. Poulouvassilis. A formal framework for ER schema transformation. In *Proceedings of ER'97*, volume 1331 of *LNCS*, pages 408–421, 1997. 96, 98
10. P.J. McBrien and A. Poulouvassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998. 96, 98, 103
11. P.J. McBrien and A. Poulouvassilis. A uniform approach to inter-model transformations. In *Advanced Information Systems Engineering, 11th International Conference CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer-Verlag, 1999. 112

12. A. Poulouvasilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998. 96, 97, 100, 103, 104
13. M.T. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for data sources. In *Proceedings of the 23rd VLDB Conference*, pages 266–275, Athens, Greece, 1997. 97
14. A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990. 96
15. M. Templeton, H. Henley, E. Maros, and D.J. Van Buer. InterViso: Dealing with the complexity of federated database access. *The VLDB Journal*, 4(2):287–317, April 1995. 97
16. M.E. Vidal, L. Raschid, and J-R. Gruser. A meta-wrapper for scaling up to multiple autonomous distributed information sources. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS98)*, pages 148–157. IEEE-CS Press, 1998. 97

A Methodology for Clustering Entity Relationship Models—A Human Information Processing Approach

Daniel L. Moody

Department of Information Systems, University of Melbourne
d.moody@dis.unimelb.edu.au

Andrew Flitman

School of Business Systems, Monash University
a.flitman@infotech.monash.edu.au

Abstract. This paper defines a method for decomposing a large data model into a hierarchy of models of manageable size. The purpose of this is to (a) improve user understanding and (b) simplify documentation and maintenance. Firstly, a set of principles is defined which prescribe the characteristics of a “good” decomposition. These principles may be used to evaluate the quality of a decomposition and to choose between alternatives. Based on these principles, a manual procedure is described which can be used by a human expert to produce a relatively optimal clustering. Finally, a genetic algorithm is described which automatically finds an optimal decomposition. A key differentiating factor between this and previous approaches is that it is soundly based on principles of human information processing—this ensures that data models are clustered in a way that can be most efficiently processed by the human mind.

1. INTRODUCTION

This paper develops a method for decomposing a large data model into a hierarchy of data models of manageable size—this is called *data model clustering* (Akoka and Comyn-Wattiau, 1996). The purpose of clustering a data model in this way is to (a) improve user understanding and (b) simplify documentation and maintenance. The research question can therefore be stated as: “How can a data model be decomposed in a way which maximises human comprehension and minimises documentation and maintenance effort?”

Levelled Data Models

A previous paper (Moody, 1997) defined a method for representing large data models based on the organisation of a street directory. A *Levelled Data Model* consists of the following components:

- A high level diagram, called the *Context Data Model*, which provides an overview of the model and how it is divided into subject areas. This corresponds to the key map in a street directory.
- A set of named *Subject Area Data Models* which show a subset of the data model (a *subject area*) in full detail. These are represented as modified Entity Relationship diagrams and correspond to detail maps in a street directory. *Foreign entities* are used to show relationships to entities on other subject areas, and correspond to inter-map references in a street directory.

- A range of *indexes*, which are used to help locate individual objects (entities, relationships and attributes) within each subject area. These correspond to street and locality indexes in a street directory.

The model may be organised into any number of levels, depending on the size of the underlying model, resulting in a hierarchy of models at increasing levels of detail. In this method, the mechanism of *aggregation* (Smith and Smith, 1978) is used to create higher level constructs (subject areas) from the primitive entities and relationships of the enterprise. This is the same mechanism used to group attributes into entities, and so is a natural extension to the Entity Relationship approach. Aggregation of attributes into entities may be called *atomic aggregation* while aggregation of entities into subject areas may be called *molecular aggregation* (Akoka and Comyn-Wattiau, 1996).

Objectives of this Paper

The objective of this paper is to develop a method for hierarchically decomposing large data models. According to Wand and Weber (1990), a comprehensive method for decomposition should be able to:

- Evaluate the quality of decompositions and choose between alternatives
- Prescribe how to generate a “good” or optimal decomposition.

Section 3 defines a set of principles for evaluating the quality of decompositions and choosing between alternatives, which satisfies requirement (a). Section 4 describes a manual procedure for generating a “good” decomposition based on these principles, while Section 5 describes an automated procedure which finds an optimal decomposition—this addresses requirement (b).

2. PREVIOUS RESEARCH

The approaches to data model clustering which have been proposed in the literature are summarised in Table 1. The table compares the methods in terms of (a) criteria used to cluster entities or *decomposition principles* (b) type of procedure (manual, automated or none) and (c) levels of clustering allowed.

Table 1. Data Model Clustering Approaches

METHOD	DECOMPOSITION PRINCIPLES	PROCEDURE	LEVELS
Martin, 1983	Functional dependencies (Level 1) Association strength (Level 2)	Manual	Two
Feldman and Miller, 1986	Functional dependencies (Level 1) Subjective (Level 2)	Manual	Two
Teory, Wie, Bolton and Koenig, 1989	Functional areas, dominant entities, levels of cohesion	Manual	Two
Batini, Ceri and Navathe, 1992	Coupling, cohesion, schema balancing, concept balancing	None	Unlimited
Francalanci and Percini, 1994	Closeness (syntactic cohesion), affinity (semantic cohesion), balancing	Automated	One
Akoka and Comyn-Wattiau, 1996	Semantic distance	Automated	One

The major deficiencies identified in the existing literature are:

- Lack of cognitive justification: A major weakness in all of the approaches reviewed is their lack of justification in cognitive theory. To be truly effective in improving human understanding, clustering approaches need to be soundly based on principles of human information processing.
- Lack of size constraints: Surprisingly, none of the approaches reviewed prescribe the size of clusters, either in terms of an “optimal” size or acceptable bounds. The stated aim of all methods is to reduce the model to parts of “manageable size”, but none of them define what this is. Leaving this to human judgement is unlikely to lead to a good solution—empirical studies show that people are unaware of their cognitive limits, and tend to include more information than can be efficiently processed (O’Reilly, 1980). There seems to be an implicit belief among humans that more information is always better, which inevitably leads to *information overload* (Moody and Walsh, 1999).
- Lack of automation: Only two approaches provide automated solutions to the problem: Francalanci and Pernici (1994) and Akoka and Comyn-Wattiau (1996). However, Francalanci and Pernici’s approach requires more effort from the user than carrying out the partitioning manually, because of the need to define semantic “affinities” between pairs of entities.
- Levels of decomposition: A basic principle of hierarchical decomposition is that any number of levels should be used, depending on the size of the underlying system (De Marco, 1978; Simon, 1982; Klir, 1985; Flood and Carson, 1993). Most of the approaches proposed are limited to a fixed number of levels of decomposition. The two automated methods allow for only *one* level of clustering, which is very restrictive. Batini et al (1992) is the only approach which allows an unlimited number of levels, but no procedure of any kind (manual or automated) is given for doing this.
- Lack of empirical testing: A final weakness of the methods reviewed is the lack of empirical research into their effectiveness in practice. It is stated and argued by the authors that their methods provide an effective solution to the problem, but these claims are unsubstantiated. Most authors demonstrate their approach using examples, but while this shows that the method “works” at some level (in that it produces a solution), it says nothing about its effectiveness in practice. That is, how do we know the results are any better in terms of user understanding or documentation and maintenance than clustering by intuition alone?

All of these deficiencies are addressed in this paper¹.

3. DECOMPOSITION PRINCIPLES

In this section, we define a set of principles for clustering data models, which prescribe the characteristics of a “good” decomposition. These provide the basis for evaluating the quality of a decomposition, and choosing between alternatives.

¹ The issue of empirical validation is only partially addressed here. We conduct some preliminary empirical testing in this paper, but more rigorous testing is proposed as part of further research.

They are also used in subsequent sections to develop methods for generating good or optimal decompositions.

Formal Statement of Problem

We formally define the problem of data model clustering as follows:

- The initial system (or *problem state*) is a data model (D), consisting of a set of entities (E) and a set of relationships between them (R). Each relationship in R defines an association between two entities in E , which may not be distinct (recursive relationships are allowed).²
- The terminal system (or *solution state*) is a hierarchy of subject areas (S), organised into a number of levels ($L_1, L_2, \dots$). Each subject area is a subset of elements at the next level down. At the lowest level (L_1), each subject area is a subset of entities in E .

This results in a hierarchy of subject areas, with successive levels representing increasing levels of abstraction (Figure 1). This is called a *multi-level structure system* (Klir, 1985) or *level structure* (Weber, 1997). Each level of the hierarchy consists of a set of elements and relationships between them. Relationships between subject areas are derived from relationships between entities: if there is a relationship in R between an entity E_i which belongs to subject area S_x and an entity E_j which belongs to subject area S_y , then a relationship will exist between S_x and S_y .

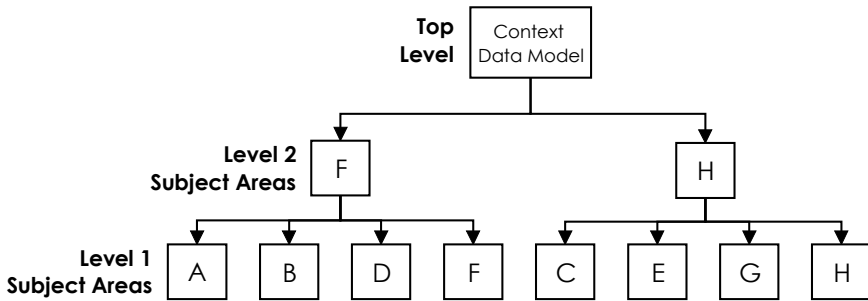


Figure 1. Multi-Level Structure System

We also make the following definitions:

- A *boundary relationship* is a relationship between entities from *different* subject areas. If a boundary relationship exists between two subject areas, they are called *adjoining subject areas*.
- An *internal relationship* is a relationship between two entities from the same subject area.

²

Ternary relationships (relationships which connect three or more entities) are not considered in this paper. Although they were part of the Entity Relationship model as originally proposed by Chen (1976), they are rarely used in practice (Hitchman, 1995). The method could be easily extended to handle such relationships if required.

Boundary relationships are shown on the Subject Area Data Models for both subject areas involved. Entities from adjoining subject areas included as part of boundary relationships are represented as *foreign entities*—these act as cross-references between subject areas.

What Makes A Good Decomposition?

There are an enormous number of alternative decompositions that can be produced for a particular problem (Klir, 1985). As the number of elements in a system increases, the number of ways in which it can be decomposed increases exponentially. According to Weber (1997), the number of possible decompositions is 2^n , where n is the number of elements in the system. This means that for an average application data model (≈ 120 entities), there are over 10^{36} different ways that it can be decomposed into subject areas. In the case of an enterprise data model (≈ 1200 entities), there are over 10^{360} possibilities!

For this reason, we need formal rules or *principles* for evaluating the quality of decompositions and choosing between alternatives. Wand and Weber (1990) argue that for information systems design to progress from an art to a science, we need to find formal ways of evaluating alternative designs rather than relying solely on the judgement of the designer. The “goodness” of a decomposition should also be measurable to reduce subjectivity and bias in the evaluation process. *Metrics* should therefore be defined for each principle.

Principle 1: Complete

This principle requires that each entity is included in *at least one* subject area. This ensures that all entities and relationships in the original model are preserved in the decomposition, so that the decomposition is *lossless* (Weber, 1997). This principle should be applied at each level of the hierarchy—that is, each Level 1 subject area must be included in at least one Level 2 subject area, each Level 2 subject area must be included in at least one Level 3 subject area, and so on.

Metric: Union of Subject Areas

This principle can be verified by ensuring that the union of subject areas at each level of the model equals the set of elements at the next level down.

Principle 2: Non-Redundant

This principle requires that each entity is included in *at most one* subject area. This ensures that subject areas form non-overlapping (disjoint) subsets of E and therefore minimises redundancy³. This simplifies maintenance by ensuring that changes to each entity can be made on a single subject area. It also simplifies user understanding and verification—overlap between subject areas can lead to problems in validation of models in practice (Moody, 1997). This principle should also be applied at each level of the hierarchy.

³ This minimises rather than eliminates redundancy between subject areas. A certain amount of redundancy will still exist between subject areas in the form of foreign entities.

Metric: Intersection of Subject Areas

This principle can be verified by ensuring that the intersection between subject areas at each level of the model is null.

Principle 3: Self Contained

This principle requires that each subject area forms a *fully connected subgraph* of the original model (D). This assists understanding by making sure that each subject area forms an integrated whole.

Metric: Fully Connected

This principle can be verified by ensuring that for any pair of entities on the same subject area, a path exists between them consisting only of internal relationships.

Principle 4: Unified

Each subject area should be named after one of the entities on the subject area, called the central entity. This helps to ensure that all entities on the subject area relate to a single concept. Identifying appropriate central entities is the key to identifying meaningful subject areas. Central entities act as “nuclei” around which other entities are clustered to form subject areas.

Central entities should be chosen as the entities of highest business importance—the “core” business concepts in the model. Of course, “business importance” is quite a subjective concept, and requires human judgement. However a useful heuristic for identifying central entities is to identify entities with the most relationships. Usually the most *highly connected* entities are also the most important entities from a business viewpoint.

As simple as this sounds, it has strong support in cognitive theory. According to *semantic network theory* (Collins and Quillian, 1969, 1972), semantic memory is structured as a network of related concepts. The concept of *spreading activation* says that nodes in a semantic network remain in a quiet state until they are activated or “primed” (Anderson and Pirolli, 1984). The activation then spreads with decreasing intensity along all pathways connected to the initial node. The level of activation decays exponentially as a function of the distance that it spreads.

Spreading activation theory predicts that recall accuracy will be highest and response latency will be lowest for concepts with large numbers of connections to other concepts, because they will receive higher levels of priming. In the case of a data model, which is structured as a network of concepts, entities involved in many relationships would therefore be more likely to be recalled. If we assume that ability to recall equates to importance, we can conclude that entities with the most relationships will be perceived as the most important. Experimental studies show that connectivity corresponds very closely to human perceptions of the business importance of entities, significant with $\alpha < .01$ (Moody and Flitman, 1999).

Metric: Connectivity

We define the *connectivity* of an entity as the number of relationships it participates in. The entities with the highest connectivities should therefore be used as central en-

tities. At higher levels of the model (Level 2 and above), central subject areas can also be identified based on their connectivities. The connectivity of a subject area is defined as the number of *boundary relationships* it has:

Connectivity (S_k) = number of relationships $r(e_i, e_j)$ where $e_i \in S_k$ and $e_j \notin S_k$

Principle 5: Cognitively Manageable

As discussed earlier, a major weakness in all methods previously proposed in the literature is that they fail to define the “manageable” size of clusters—either in terms of an optimal size or acceptable bounds. We argue that in order to be conceptually manageable, subject areas should be strictly limited in size. We also argue that this limit should be based on the limits of human cognitive capacity.

There is universal agreement among cognitive psychologists that, due to limits on short term memory, the human mind can only handle “seven plus or minus two” concepts at a time (Miller, 1956; Baddeley, 1994). Once the amount of information exceeds these limits, it must be organised into larger and larger chunks, each containing more information and less detail (Uhr et al, 1962). We therefore define the *maximum size* of subject areas as the upper limit of human cognitive capacity (nine concepts). This should be used as the limit at all levels of the model, to ensure that each subject area forms a cognitively “digestible” unit of information.

Limiting the size of subject areas helps to overcome both the limitations of the human mind in dealing with large amounts of information (*understanding*) and the restrictions of physical sheets of paper (*documentation and maintenance*). If nine concepts is used as the limit at all levels, diagrams can be easily drawn on standard sized paper, and the need for reduced fonts and/or crossed lines is virtually eliminated.

Metric: Maximum Size of Subject Areas

The *size* of a subject area is defined as the number of distinct concepts it contains (which may be either entities or subject areas).

Principle 6: Flexible

Systems need to adapt to changes over time, and should therefore be organised in a way that is resilient to change (Davis and Olson, 1985; Wand and Weber, 1990; Simon, 1982). Data models tend to grow in size over time, as new requirements are added or systems expand in scope. The partitioning of the data model into subject areas should therefore allow flexibility for growth. A data model which consists of subject areas that are all of the maximum size (nine) will have to be repartitioned if even a single entity is added.

Metric 1: Capacity for Growth

Flexibility can be measured by calculating the percentage of growth possible before the model needs to be re-partitioned. This is defined as:

$((\text{No. of Level 1 subject areas} * 9) - (\text{No. of entities in E})) * 100 / (\text{No. of entities in E})$

As a rule of thumb, at least 20% growth should be allowed.

Metric 2: Optimal Size of Subject Areas

Capacity for growth is a *bottom-up* measure—it can only be used to evaluate a decomposition after the event, and cannot be used in a top-down manner to guide the decomposition process. A more direct way to ensure flexibility is to prescribe a standard or “optimal” size for subject areas which allows sufficient room for expansion. The model can then be partitioned in such a way that the *average* size of subject areas is as close as possible to the optimal.

We define the optimal size of subject areas as seven. This allows for growth of two entities per subject area, or about 30%. There is also strong cognitive justification for using seven as the optimal number of concepts for each subject area. Recall that the limits on short-term memory are defined as a range—“seven, plus or minus two”. This means that some people will have a limit of five concepts, others will have a limit of nine concepts, while most people will be around seven. To maximise understandability to all people, it is preferable to use the midpoint rather than the upper limit of human cognitive capacity as the average size of clusters.

Optimal Number of Subject Areas

The optimal size of subject areas can be used to calculate the *optimal number of subject areas* for each level of decomposition. This can be used to guide the decomposition process from the beginning. The optimal number of subject areas at Level n is defined as the number of entities in E divided by 7^n , rounded to the nearest whole number. For example, for a model with 125 entities, the optimal number of subject areas at Level 2 will be 3 ($125/49 = 2.55$).

Optimal Number of Decomposition Levels

The optimal size of subject areas can also be used to calculate the *optimal number of levels* of decomposition. This is defined as the number of entities in E log to base 7, rounded down to the next lowest whole number. For example, for a model with 125 entities, two levels will be required ($125 \log 7 = 2.48$).

Principle 7: Balanced

Another characteristic of a good decomposition is the principle of *equal abstraction* or *balancing* (De Marco, 1978; Klir, 1985; Francalanci and Pernici, 1994). This states that each subsystem should be approximately equal in size.

Metric 1: Standard Deviation in Subject Area Size

Balancing can be measured by calculating the standard deviation in the size of subject areas, following Francalanci and Pernici (1994).

Metric 2: Minimum Size of Subject Areas

Standard deviation in size is a *bottom-up* measure—it can only be used to evaluate a decomposition after the event, and cannot be used to guide the decomposition process. A simpler and more direct way to achieve an acceptable level of balancing is to define a *minimum size* for subject areas. We define the minimum size of subject areas as five concepts—this allows a deviation of 30% in both directions from the average size (given the maximum size of subject areas defined by Principle 5 and the average

size of subject areas defined by Principle 6). Note that restricting the bounds further to between six and eight would not work, because for data models of a certain size (e.g. 17 entities) no solution could be found which would satisfy this principle.

As a result of Principles 5, 6 and 7, we have now defined a minimum, maximum and optimal size for clusters. Subject areas at all levels should consist of a minimum of five, a maximum of nine and an average of seven elements. This is literally *seven plus or minus two* concepts per subject area. There is a clear symmetry in using the limits of human cognitive capacity as the limits on the size of subject areas. Grouping of entities into subject areas is done to facilitate “chunking” in the human mind. For this to be most effective, clustering should mirror as closely as possible the way concepts are chunked in the human mind.

Principle 8: Loosely Coupled

Coupling is defined as the strength of interconnections *between* different subsystems (*inter-molecular forces*). Coupling is widely accepted to be one of the most important measures of the quality of a decomposition and should be minimised to increase the independence of the parts of the system (Alexander, 1964; Simon, 1982). Systems that are loosely coupled are generally easier to maintain because subsystems can be maintained relatively independently of each other (Yourdon and Constantine, 1979; Davis and Olson, 1985; Weber, 1997; Flood and Carson, 1993). Reducing coupling also reduces the overall complexity of the system—according to complexity theory, the complexity of any system is defined by the number of components and the number of relationships between them (Klir, 1985; Pippenger, 1978).

In the case of a Levelled Data Model, coupling corresponds to the number and strength of relationships between subject areas (boundary relationships). Minimising boundary relationships:

- Improves understanding by ensuring that subject areas can be understood independently of each other and reducing the need to navigate between subject areas
- Simplifies documentation by reducing the need to show cross-references between subject areas (via *foreign entities*)
- Simplifies maintenance by ensuring that subject areas can be maintained relatively independently and by minimising redundancy between them (by reducing the number of foreign entities)

Metric 1: Number of Boundary Relationships

The simplest measure of the *coupling* of a decomposition is the number of relationships between subject areas (*boundary relationships*).

Metric 2: Sum of Boundary Relationships Strengths

A finer resolution level measure of coupling can be obtained by assigning different weights to different types of relationships to indicate their relative semantic strength (following Akoka and Comyn-Wattiau, 1996). The coupling of the decomposition can then be calculated as the sum of the strengths of boundary relationships.

Principle 9: Highly Cohesive

The complementary concept to coupling is *cohesion*, which is defined as the strength of associations *within* subsystems (*intra-molecular* forces). Cohesion should be *maximised* in order to increase independence of subsystems. Subsystems which are highly cohesive are likely to be more independent of each other, which simplifies maintenance (Yourdon and Constantine, 1978; Flood and Carson, 1993). Also, subsystems that are highly cohesive will be easier to understand because they can be encoded as a single integrated “chunk” of information rather than a number of relatively independent “chunks” (Eysenck and Keane, 1992; Weber, 1997).

Metric 1: Number of Internal Relationships

The simplest measure of the cohesion of a decomposition is the number of relationships within subject areas (*internal relationships*).

Metric 2: Sum of Internal Relationships Strengths

As with coupling, a finer resolution measure of cohesion can be obtained by assigning different weights to different types of relationships to indicate semantic “strength”. The cohesion of the decomposition can then be calculated as the sum of the strengths of internal relationships. Attempts to define measures of cohesion based on perceived semantic similarity (e.g. Francalanci and Pernici’s concept of “affinity”) are highly subjective and very time consuming to apply in practice.

Coupling vs Cohesion

Note that the sum of the values for cohesion and coupling for a decomposition will always equal the total number of relationships in the model (or the sum of their strengths, depending on the metric used). As a result, increasing coupling will always decrease cohesion by an identical amount. This means that these two principles are *logically dependent*, and, following the rule of parsimony, we can eliminate one of them. Alternatively, we can combine them together and create a new metric called *relative cohesion*, which is the ratio of cohesion to coupling. This provides a way of comparing the quality of decompositions independent of the size of the underlying data model. As a general rule, relative cohesion should be greater than or equal to two—this ensures that associations *within* subject areas (*intra-molecular* forces) are at least twice as strong as those *between* subject areas (*inter-molecular* forces).

Summary of Principles

The principles may be classified into two categories:

- Constraints: these are mandatory requirements of a decomposition, and can be used to differentiate between *valid* and *invalid* solutions.
- Objectives: these define the desirable characteristics of a decomposition, and can be used to choose between a number of valid solutions.

The purpose of the decomposition process will be to *optimise objectives* while *satisfying constraints*. The problem of data model decomposition can therefore be formulated as an *optimisation problem*. The principles are summarised in Table 2 below. Note that Principles 6 and 7 may be considered either as constraints or objectives de-

pending on the metric used. Principles 8 and 9 have been combined together because they were logically equivalent.

Table 2. Summary of Principles and Metrics

PRINCIPLE	TYPE	METRICS
1. Complete	Constraint	Union of subject areas = E
2. Non-redundant	Constraint	Intersection of subject areas = $\emptyset$
3. Self-contained	Constraint	Fully connected
4. Unified	Constraint	Central entities = entities with maximum connectivity
5. Cognitively manageable	Constraint	Maximum size of subject areas = 9
6. Flexible	Constraint Objective (as close as possible to)	Capacity for growth $\geq 20\%$ Average size of subject areas $\rightarrow 7$
7. Balanced	Objective (minimise) Constraint	Standard deviation in subject area size Minimum size of subject areas = 5
8. Coupling/ Cohesion	Objective (maximise)	Relative cohesion = $n(\text{internal relationships})/n(\text{boundary relationships}) \geq 2$

4. MANUAL CLUSTERING PROCEDURE

In this section, we define a manual procedure for clustering a data model based on the principles defined in Section 3. While the principles defined are sufficient on their own for a human to carry out the task, a procedure simplifies the task by breaking it down into manageable steps. This reduces the conceptual difficulty of the task and should therefore improve task performance (Flood and Carson, 1993).

Step 1: Identify Central Entities

The first step of the clustering process is to identify central entities, which will form the “nuclei” for subject areas. For a model with n entities, $n/7$ central entities will be required to allow sufficient room for growth [Principle 6]. The entities of greatest business significance should be chosen as central entities [Principle 4]. These can be identified based on their connectivities, although user input may also be used.

Step 2: Cluster Entities Around Central Entities

Other entities should then be clustered around each of the central entities, making sure that each entity is assigned to one and only one subject area [Principles 1 and 2]. Each subject area should consist of a minimum of five and a maximum of nine entities [Principles 5 and 7]. The upper bound ensures that subject areas are within the limits of human cognitive capacity, while the lower bound helps to achieve balancing between the size of clusters. Experience shows that Principle 3 does not need to be explicitly included in this step, because it is implicit for humans performing the task.

Step 3: Evaluate and Refine Partitioning

Calculate the number of internal relationships (*cohesion*) and the number of boundary relationships (*coupling*) for the partitioning. The level of cohesion should be at least twice the level of coupling [Principle 8]. Alternative ways of clustering entities should be explored to try to reduce the number of boundary relationships.

Step 4: Higher Level Partitioning

If at the end of this process, the Context Data Model contains more than nine subject areas, it will need to be decomposed as well. The “seven plus or minus two” rule must be applied at all levels of the model to ensure that complexity is manageable at each stage. Steps 1, 2 and 3 should then be applied to the Context Data Model.

- The number of Level 1 subject areas should be divided by seven to determine the number of Level 2 subject areas required [Principle 6]. The $n/7$ most important Level 1 subject areas should then be identified based on their connectivities [Principle 4].
- Clustering of Level 1 subject areas into Level 2 subject areas can then be done in a relatively deterministic way. The objective should be to minimise coupling [Principle 8] while obeying the seven, plus or minus two rule [Principles 5 and 7]. At the end of this step, each Level 1 subject area should be assigned to one and only one Level 2 subject area [Principles 1,2].

This process should be repeated until there is nine concepts or less at the top level.

5. AUTOMATIC CLUSTERING PROCEDURE

In this section, we describe a genetic algorithm for automatically clustering a data model based on the principles defined in Section 3. Because of the enormous number of decompositions that are possible in even small data models, it is clearly beyond human cognitive capacity to find an optimal solution. According to the principle of *bounded rationality*, humans will only explore a limited range of alternatives and consider a subset of the decomposition principles in order to make the task cognitively manageable (Simon, 1982).

Genetic Algorithms

Given the number of alternative decompositions that are possible, combined with the range of principles we need to satisfy, a robust technique capable of solving complex non-linear systems is required. Since the robustness of traditional optimisation techniques has been frequently questioned, we used a genetic algorithm to solve the problem. *Genetic algorithms* differ from standard optimisation techniques in four ways (Goldberg, 1989):

1. They work with a coding of the parameter set, not the parameters themselves
2. They search from a population of points
3. They use payoff information, not derivatives
4. They use probabilistic transition rules, not deterministic rules.

Genetic algorithms are a computer simulation of genetic theory. Each *generation* consists of a population of individuals, represented by *chromosomes*, which have varying

levels of *fitness*. After the fitness of each member is determined, the next generation is created. The chances of an individual surviving to the next generation are proportional to its fitness. *Random mutations* and *crossovers* between population members also affect each generation. The likelihood of mutations and crossovers are determined by the mutation and crossover rate respectively. After each generation, the population will tend toward “fitter” members and therefore a more optimal solution.

Optimisation Algorithm

We used a non-repeating enumerated chromosome for this problem. This is simply a chromosome consisting of an ordered string of numbers. The numbers used represent either entities or subject areas. In our case the numbers were the integers 1 to n (where n represents the number of entities in the initial data model), plus integers $n+1$, $n+2$, ..., $n+m-1$ where m is the number of subject areas. The process of mutation and crossover was modified to take account of the special non-repeating and order dependent characteristics of our chromosome.

A population of 32-bit length ‘chromosomes’ was initialised randomly. The “fitness” of each chromosome was then calculated based on the principles and metrics defined in Section 3. The probability of survival to the next generation for each chromosome was proportional to its fitness. The surviving chromosomes in the population were combined via single point crossover and a number of bits mutated in each generation.

The pseudo-code used for the genetic algorithm optimisation is shown below (full details of the algorithm can be found in Moody and Flitman, 1999).

```
CLUSTERING ALGORITHM (Initial System, Final System)
Generate random population
DO for all possible subject counts
    DO WHILE populations since optimum chromosome < 200
        Determine population fitness
        Make probability of survival proportional to fitness
        Recombine via single point crossover and mutation
    END WHILE
END WHILE
```

Higher Level Clustering

Note that the above algorithm results in only a single level of clustering. Given a set of entities E and a set of relationships R , it will group the entities into a set of Level 1 subject areas. The need to organise a model into multiple levels can be satisfied by repeated application of the algorithm, using the outputs of one cycle as the input to the next. For example, if there are more than nine Level 1 subject areas as a result of the first decomposition cycle, the set of Level 1 subject areas will be used as input to the algorithm to generate a set of Level 2 subject areas. This can be repeated as many times as necessary.

We define *Current_System* as the set of elements that need to be clustered at a particular stage of decomposition. When we begin, *Current_System* is the set of entities E . After the first pass through the algorithm, *Current_System* becomes the set of Level 1 subject areas. If there are more than nine of these, then the algorithm is invoked again, resulting in a set of Level 2 subject areas. This process continues until *Current_System* consists of nine or less elements.

The pseudo code for this is:

```

Current_System := E
Number of Levels = 0
DO UNTIL Clustering_Complete
  IF Number of elements in Current_System ≤ 9
    Clustering_Complete := True
  ELSE
    Perform CLUSTERING ALGORITHM (Current_System, Result)
    Number of Levels := Number of Levels + 1
    Current_System := Result
  END IF
END DO

```

Empirical Results

The algorithm was coded in Delphi 3 and tested on a range of data models of different sizes.

Parameter Settings

Choosing optimal parameters for the genetic algorithm optimisation is not straightforward. Schaffer et al (1989) provide an extensive study of parameter settings for a suite of functions and found that good on-line performance can be achieved with:

- Population size: 20-30
- Crossover rate: 0.75-0.95
- Mutation rate: 0.005-0.01

We tested their results in our study together with other combinations of parameters. We found that the optimal parameters for the data model decomposition problem were:

- Population size: 200
- Crossover rate: 0.90
- Mutation rate: 0.01

Efficiency

For a data model with 22 entities, the system took 68 “generations” to find the optimal result. A total of 282 “generations” were produced in all, which took 220 seconds on a Pentium 233Mhz PC. For a data model with 88 entities, the system took 120 “generations” to find the optimal result. A total of 320 “generations” were produced in all, and took 385 seconds. These timeframes are clearly acceptable for use in practice, and are considerably less than the time a human expert took to complete the same tasks.

Effectiveness

On a range of data models, the algorithm consistently outperformed a human expert in terms of the metrics defined in Table 2. For the 22-entity model, a decrease in standard deviation of subject area size (balancing) of 62% and a decrease in relative cohesion of 20% was achieved. For the 88-entity model, a decrease in standard deviation of subject area size (balancing) of 94% and a decrease in relative cohesion of 34% was achieved. The improvement in performance increased with the size of the model, which was consistent with *a priori* expectations—the number of alternatives and therefore the complexity of the task increases with data model size, which will tend to increase the disparity between human and computer processing capabilities.

6. CONCLUSION

This paper has described a method for clustering large data models which is both practical to apply and consistent with known principles of human information processing. It has defined:

- A set of principles and metrics for evaluating the quality of a decomposition and choosing between alternatives (Section 3)
- A manual procedure for decomposing a model based on these principles, which enables a human designer to produce a relatively optimal solution (Section 4)
- A genetic algorithm that automatically finds an optimal decomposition based on the principles defined (Section 5).

Practical Significance

The complexity of data models is a major barrier to the effective communication of data models in practice. Data model clustering provides a solution to this problem by dividing a large data model into conceptually “bite-sized” pieces. This paper assists practitioners in carrying out the clustering task in the following ways:

- The development of formal principles to guide the clustering process reduces cognitive uncertainty and improves consistency between different analysts performing the task.
- The use of a step-by-step procedure for carrying out the decomposition reduces the conceptual difficulty of the task (*efficiency*) and should therefore improve task performance (*effectiveness*).
- The use of an automated clustering algorithm reduces time and effort required (*efficiency*) and results in a better solution than would be possible using a manual procedure (*effectiveness*).

Theoretical Significance

A number of approaches for clustering data models have been previously published in the literature. This paper addresses the deficiencies identified in the existing literature identified in Section 2:

- It defines a set of clustering principles which are soundly based on principles of human information processing. The use of psychology as a reference discipline provides the basis for a more effective solution to the problem than previous approaches, which have drawn mainly on the software engineering literature. Software engineering is concerned with constructing efficient and reliable systems, whereas data model clustering is about presenting information in a way which the human mind can efficiently process.
- It defines the concept of “manageable size” for clusters. It defines an optimal size for subject areas as well as upper and lower bounds. These limits are based on the limits of human cognitive capacity. Definition of these limits allows *a priori* calculation of the optimal number of levels and number of clusters at each level, which previously has been left to human judgement.
- It provides a fully automated method for finding an optimal clustering, as well as a manual procedure for doing so. Only two of the previous approaches have

provided an automated solution to the problem, and none of the previous approaches have provided both.

- It allows multiple levels of decomposition, depending on the size of the underlying model. All optimisation algorithms proposed in the literature have been limited to a single level. Only one other approach allows for an unlimited number of levels (Batini et al, 1992), but no procedure was specified for doing this.

Further Research

We have argued on theoretical grounds that this method offers a better prospect of success in clustering large data models than previous approaches proposed in the literature. While theoretical justification is important, ultimately the soundness of any method is an empirical rather than a theoretical question (Ivori, 1986). The lack of empirical testing is one of the most serious deficiencies in the existing literature.

Some limited empirical testing of the method has been carried out as part of this paper. However it is the task of further research to systematically evaluate the effectiveness of the method in practice. An experiment has been designed to compare the quality of decompositions produced by (a) a human expert using judgement alone, (b) a human expert following the principles defined in Section 3, (c) a human expert following the procedure described in Section 4, (d) the genetic algorithm described in Section 5. The results will be evaluated by a panel composed of experienced data modellers and business users. Business users will be asked to evaluate the understandability of the different decompositions, while the data modellers will be asked to evaluate ease of documentation and maintenance.

REFERENCES

1. AKOKA, J. AND COMYN-WATTIAU, I. (1996) "Entity Relationship and Object Oriented Model Automatic Clustering", *Data and Knowledge Engineering*, 20.
2. ALEXANDER, C. (1968): *Notes on the Synthesis of Form*, Harvard University Press, Boston.
3. ANDERSON, J.R. and PIROLI, P.L. (1984): "Spread of Activation", *Journal of Experimental Psychology: Learning, Memory and Cognition*, 10, 4.
4. BADDELEY, A. (1994): "The Magical Number Seven: Still Magic After All These Years?", *Psychological Review*, 101, 2.
5. BATINI, C., CERI, S. AND NAVATHE, S.B. (1992) *Conceptual Database Design: An Entity Relationship Approach*, Benjamin Cummings, Redwood City, California.
6. COLLINS, A.M. and QUILLIAN, M.R. (1969): "Retrieval Time from Semantic Memory", *Journal of Verbal Learning and Verbal Behaviour*, 8.
7. COLLINS, A.M. and QUILLIAN, M.R. (1972): "How to Make a Language User", in *Organisation and Memory*, E. Tulving and M. Donaldson (ed.s), Academic Press, New York.
8. DAVIS, G.B. and OLSEN, M.H. (1985): *Management Information Systems: Conceptual Foundations, Structure and Development*, McGraw-Hill.
9. DE MARCO, T. (1978): *Structured Analysis and System Specification*, Yourdon Press, 1978.
10. EYSENCK, M.W. AND KEANE, M.T. (1992): *Cognitive Psychology: A Student's Handbook*, Lawrence Erlbaum Associates, Hove and London.
11. FELDMAN, P. and MILLER, D., (1986): Entity Model Clustering: Structuring a Data Model by Abstraction, *The Computer Journal*, Vol. 29, No. 4.

12. FLOOD, R.L. and CARSON, E.R. (1993): *Dealing With Complexity: An Introduction to the Theory and Application of Systems Science*, Plenum Press.
13. FRANCALANCI, C. AND PERNICI, B. (1994): "Abstraction Levels for Entity Relationship Schemas", in P. LOUCOPOULOS (ed.) *Proceedings of the Thirteenth International Conference on the Entity Relationship Approach*, Manchester, December 14-17.
14. GOLDBERG, D. (1989): "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, p15-21.
15. IVARI, J. (1986): Dimensions Of Information Systems Design: A Framework For A Long Range Research Program. *Information Systems*, June, 39-42.
16. KLIR, G.J. (1985): *Architecture of Systems Problem Solving*, Plenum Press, New York.
17. MARTIN, J. (1983): *Strategic Data Planning Methodologies*, Prentice-Hall.
18. MILLER, G. (1956): The Magical Number Seven, Plus Or Minus Two: Some Limits On Our Capacity For Processing Information, *The Psychological Review*, March.
19. MOODY, D.L. and FLITMAN, A. (1999): "Principles. Metrics and an Algorithm for Clustering Entity Relationship Models", *Department of Information Systems Working Paper*, University of Melbourne, Parkville, Victoria, Australia.
20. MOODY, D.L. and WALSH, P.A., (1999) "Measuring the Value of Information: An Asset Valuation Approach", *Proceedings of the Seventh European Conference on Information Systems (ECIS '99)*, Copenhagen, Denmark, June 23-25.
21. MOODY, D.L., "A Multi-Level Architecture for Representing Enterprise Data Models", *Proceedings of the Sixteenth International Conference on Conceptual Modelling (ER'97)*, Los Angeles, November 1-3, 1997.
22. PIPPENGER, N. (1978): Complexity Theory, *Scientific American*, 238(6): 1-15.
23. O'REILLY, C.A. (1980): Individuals and Information Overload in Organisations: Is More Necessarily Better?, *Academy of Management Journal*, Vol 23, No. 4.
24. SCHAFFER D., CARUANA R., ESHELMAN L., RAJARSHI D. (1989): "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization", *Proceedings Of The 3rd International Conference on Genetic Algorithms*, 1989, p.51-61, Morgan Kaufmann
25. SIMON, H.A. (1982): *Sciences of the Artificial*, MIT Press.
26. SMITH, J.M. and SMITH, D.C.P. (1977): Database Abstractions: Aggregation and Generalization, *ACM Transactions on Database Systems*, Vol. 2 No. 2.
27. TEORY, T.J., WEI, G., BOLTON, D.L., and KOENIG, J.A. (1989): ER Model Clustering as an Aid for User Communication and Documentation in Database Design, *Communications of the ACM*, August.
28. UHR, L., VOSSIER, C., and WEMAN, J. (1962): Pattern Recognition over Distortions by Human Subjects and a Computer Model of Human Form Perception, *Journal of Experimental Psychology*, 63.
29. WAND, Y. and WEBER, R.A. (1990): A Model for Systems Decomposition, in J.I. DeGross, M. Alavi, and H. Oppelland (Ed.s), *Proceedings of the Eleventh International Conference on Information Systems*, Copenhagen, Denmark, December.
30. WEBER, R.A. (1997): *Ontological Foundations of Information Systems*, Coopers and Lybrand Accounting Research Methodology Monograph No. 4, Coopers and Lybrand Australia, Melbourne, Australia.
31. YOURDON, E. and CONSTANTINE, L.L. (1979): "Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design", Prentice-Hall, Englewood Cliffs, NJ.

Designing Good Semi-structured Databases

Sin Yeung Lee¹ Mong Li Lee¹ Tok Wang Ling¹ Leonid A. Kalinichenko²

¹ School of Computing
National University of Singapore
{jlee, leeml, lingtw}@comp.nus.edu.sg

² Institute for Problems of Informatics
Russian Academy of Sciences
leonidk@synth.ipi.ac.ru

Abstract. Semi-structured data has become prevalent with the growth of the Internet and other on-line information repositories. Many organizational databases are presented on the web as semi-structured data. Designing a “good” semi-structured database is increasingly crucial to prevent data redundancy, inconsistency and updating anomalies. In this paper, we define a semi-structured schema graph and identify the various anomalies that may occur in the graph. A normal form for semi-structured schema graph, S3-NF, is proposed. We present two approaches to design S3-NF database, namely, restructuring by decomposition and the ER approach. The first approach consists of a set of rules to decompose a semi-structured schema graph into S3-NF. The second approach uses the ER model to remove anomalies at the semantic level.

1 Introduction

The growth of the Internet and other on-line information repositories has greatly simplified the access to numerous sources of information/data, especially through the World Wide Web. The data is presented in various forms: At one extreme we find data coming from traditional relational/object-oriented databases, with a completely known structure. At the other extreme we have data which is fully unstructured, eg images, sounds and raw text. But most of the data fall somewhere in between (semi-structured) for a variety of reasons: the data may be structured, but the structure is not known to the user; the user may know the structure, but chooses to ignore it for browsing purposes. Examples of semi-structured data include HTML documents where the structure is imposed by tags, and bibliography files where some structure is imposed by fields such as the author and the title to an otherwise unstructured text file. Note that the tags and fields are optional.

The nature of semi-structured data is fundamentally different from data in traditional databases and hence raises many new issues. Some of the common scenarios involve extracting data from the diverse information repositories across the Internet, and integrating the data from heterogeneous sources. These tasks are made more difficult because we have only a partial knowledge of the structure and that the structure is potentially “deeply nested” or even cyclic. Many researchers have proposed semi-structured data models, databases, and languages to model, store and query the World Wide Web data [1, 5, 7, 18, 15]. These works use some graph or tree models [5, 18] which provide a flexible representation of

data coming from arbitrary heterogeneous sources. Proposed query languages are closely tied to these data models.

Unfortunately, there is no notion of a precise or explicit schema in all these semi-structured databases. All the schematic information is embedded in the graphs which may change dynamically. The lack of a schema poses two problems. First, it is difficult for a user to formulate a meaningful query on the semi-structured data without any knowledge of how the data is organized. Second, it is difficult for the query processor to generate efficient plans for queries on the semi-structured data without any schema to guide it. As a result, [8] introduces DataGuides which dynamically generate and maintain structural summaries of semi-structured databases. The Dataguides are used in the Lore DBMS [15] for the user to carry out structure browsing and query formulation, as well as for the query processor to optimize query execution. More work to discover schemas from semi-structured data can be found in [21, 16, 4] and others.

However, to date, we observe that the concept of a *well-formed* or a *normal form* semi-structured schema has never been considered. This has come to our attention because data redundancy and data inconsistency may occur in a semi-structured database if the schema is not designed properly. In a traditional database, redundancy has inevitably caused updating anomalies [6]. A well-established technique to remove undesirable updating anomalies and data redundancy from relations and nested relations is normalization [13, 20, 14, 17, 19, 12].

In this paper, we will focus on how to design good semi-structured databases. We assume that we can extract the schema from a semi-structured data source or database. We will define the concept of a semi-structured schema graph and investigate the various anomalies that may appear in this model. A normal form for semi-structured schema graph, S3-NF, is proposed. Our normal form not only deals with functional dependencies and multivalued dependencies, but also removes identified anomalies from the semi-structured schema graph. Two approaches to designing S3-NF databases are given, namely, the restructuring approach and the Entity-Relationship (ER) approach. The former consists of a set of rules to decompose a semi-structured schema graph into S3-NF while the latter uses the ER model to remove the anomalies and redundancies at the semantic level. We envisage the growing importance of well designed semi-structured databases due to the increasing popularity of XML for data representation on the Web [3].

The rest of the paper is organized as follows. Section 2 gives the definitions of various concepts in a semi-structured schema graph. The anomalies that may occur in the graph is also presented. In section 3, a normal form for semi-structured schema graph, S3-NF, is defined. We also show how we can restructure a semi-structured schema graph to obtain S3-NF which does not have the undesirable anomalies. Section 4 discusses the ER approach to designing a S3-NF database. Finally, we conclude in Section 5.

2 Semi-Structured Graph: Definition and Anomalies

Data modeling people have long noticed the fact that if the database attributes are fixed in structure, the modeling power of the data model is greatly reduced. With the introduction of XML, semi-structured data model becomes widespread. However, the problem of anomalies, which was well solved for various relational models, including the flat relation model [2, 11] and the nested relation model [12], will appear again. In this section, we define the concept of Semi-Structured Schema Graph and discuss various anomalies that may appear in this model.

2.1 Motivating Example

We use the Object Exchange Model (OEM) [18] adopted in Stanford's Lore DBMS [15] to represent semi-structured data. OEM is self-describing as each object contains its own schema and there is no distinction between schema and data. Each object in OEM has an object identifier (oid) and a value. The value is either atomic, (integer, real, string, gif, html, audio, etc) or complex, that is, a set of object references denoted as a set of (label, oid) pairs. The labels are taken from the atomic type string. We can visualize OEM as a labeled graph in which vertices correspond to objects and edges represent the object-subobject relationship. Each edge has a label describing the precise nature of the relationship. Based on the OEM, Lorel [1] uses the familiar *select-from-where* syntax and path expressions to traverse the semi-structured data. For example, the path expression *Student.Course* specifies the Course subobjects of object Student.

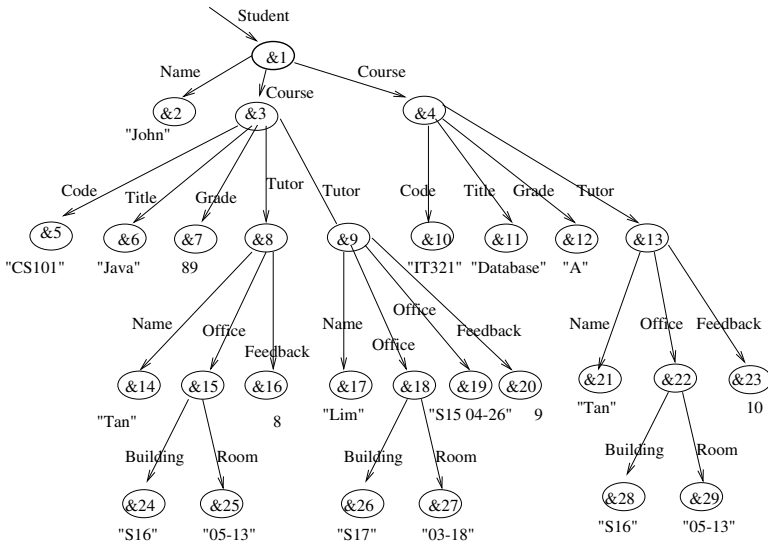


Fig. 1. Example of an OEM graph

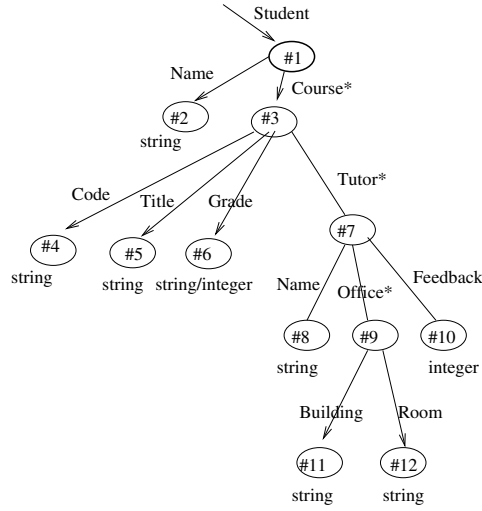


Fig. 2. Schema of Figure 1

Figure 1 shows an example of an OEM graph. A student (with attribute Name) can be enrolled in many courses (with attributes Code and Title). The student's grade for a course is kept in the attribute Grade. A tutor, with attributes Name and Office, can teach more than one course. A student can have more than one tutor for a course. The student's evaluation of a course tutor is kept in the attribute Feedback. There is no fixed or regular structure in the graph

1. a student may take zero, one or more courses
2. a course may have one or more tutors
3. a tutor may have one or more offices
4. a student's course grade may be in marks (0-100) or in grades (A to F)
5. a tutor's office is a complex structure consisting of building and room

The associated schema is shown in Figure 2.

The semi-structured database in Figure 1 is not well-designed because it contains data redundancy. The code and title of a course will be stored as many times as the number of students taking the course. Similarly, information of a tutor such as his/her office and age will also be duplicated since a tutor can teach more than one course and more than one student. Such data redundancies can be removed if we have links, denoted by dashed edges, to Course and Tutor objects as shown in Figure 3. The associated schema is shown in Figure 4.

2.2 Definition of Semi-Structured Schema Graph (S3-Graph)

Definition 1. A *Semi-Structured Schema Graph (S3-Graph)* is a directed graph defined as follows,

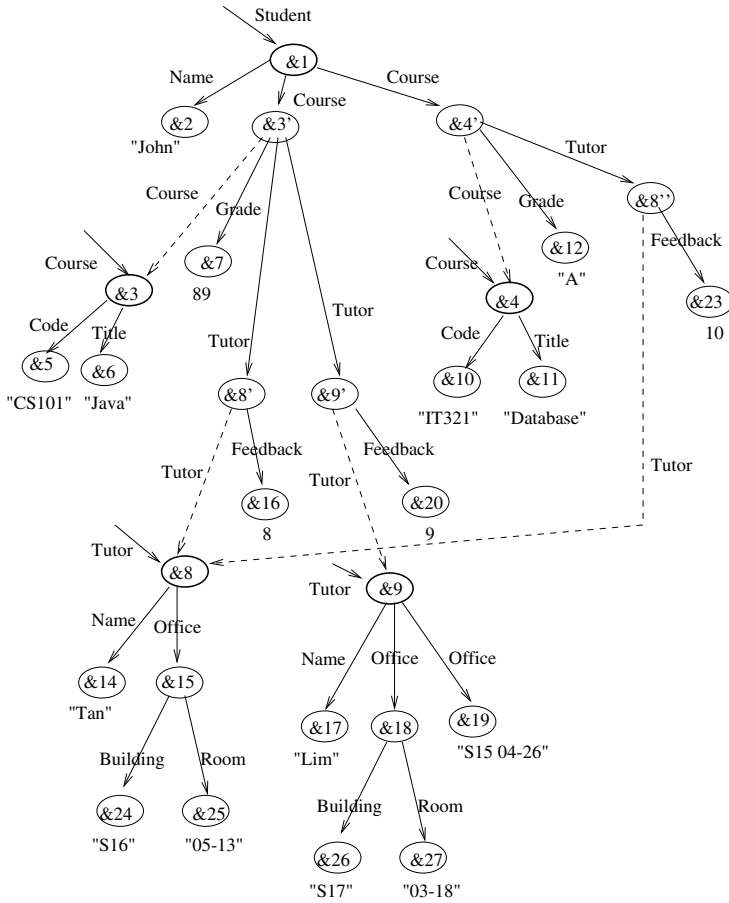


Fig. 3. A Well-Designed Semi-Structured Database

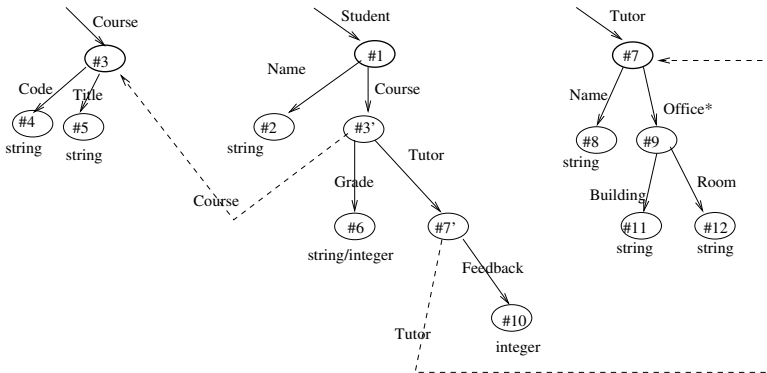


Fig. 4. Schema of the Well-Designed Semi-Structured Database in Figure 3

Each node of the graph can be classified into one of the following types:

1. *entity node* is a node which represents an entity. This entity can be of basic atomic data type such as string, date or complex object type such as student. If the entity node represents a basic atomic data type instead of a complex object type, then the entity node is also known as *leaf entity node*. Intuitively, a leaf node does not have any child (defined later) in the S3-Graph. We further attach the atomic data type as a label to the corresponding leaf entity node in the S3-Graph.
2. *reference node* is a node which references to another entity node.

Each directed edge in the graph is associated with a *tag*. The tag represents the relationship between the source node and the destination node. The tag may be suffixed with a “*”. The interpretations of tag and the suffix depend on the type of edge. There are three types of edges:

1. Component Edge

A node V_1 is connected to another node V_2 via a *component edge* with a tag T if V_2 is a component of V_1 . We represent this edge by a solid arrow line. If T is suffixed with a “*”, the relationship is interpreted as “The entity type represented by V_1 has many T ”. Otherwise, the relationship is interpreted as “The entity type represented by V_1 has at most one T ”.

2. Referencing Edge

A node V_1 is connected to another node V_2 via a *referencing edge* if V_1 references the entity represented by node V_2 . We represent this edge by a dashed arrow line. In this case, the relationship is interpreted as “ V_1 references V_2 ”.

3. Root Edge

A node V_1 is pointed by a *root edge* with a tag T if the entity type represented by V_1 is owned by the database. We represent this edge by a solid arrow line without any source node for the edge. The tag T is not suffixed. In this case, the edge is interpreted as “The database has many T ”. Furthermore, we shall call V_1 as a *root node* in the S3-Graph.

Finally, some roles R can be associated with a node V if there is a directed (*component or referencing*) edge pointing to V with tag R after removing any suffix “*”.

Example 1. In Figure 4, node #1 represents an entity node, which represents the entity STUDENT. This is also one of the root nodes. This node is associated with the role “Student”. Node #2 is another entity node of which database instance holds a string representing the NAME of a student. It is associated with the role “Name”. It is also a leaf node associated the atomic data type “string”. Hence any “NAME” data is of string type. The directed edge between node #1 and node #2 represents “Each STUDENT has at most one NAME”.

Node #3’ is a reference node. It references the entity node which represents COURSE. In this case, #3’ represents the same entity as in #3. The edge connecting #1 and #3’ is interpreted as “Each STUDENT has many COURSES”.

Note that in a more complex example, a node can be associated with more than two roles. For example, a PERSON can be both a LAB-MEMBER as well as a LAB-SUPERVISOR.

Definition 2. Given a S3-Graph G , if a node P of role R_P is connected via a component edge to another node C of role R_C , then R_P is referred to as a *strongly connected parent* of R_C , and R_C as a *strongly connected child* of R_P .

Similarly, if there is a path of only component edges which connects a node A of role R_A to a node D of role R_D , then R_A is said to be a *strongly connected ancestor* of R_D . If R_A is a strongly connected ancestor of R_D , then R_D is one of the *strongly connected descendants* of R_A .

Example 2. Refer to Figure 4, Student is a strongly connected parent of Tutor, and Office is a strongly connected descendent of Tutor. However, Office is not a strongly connected descendent of Student because any path that connects a node of Student role to a node of Office role must go through a referencing edge between node #7 and node #7'.

In the above example, we see that strongly connected ancestor is not necessarily transitive. In fact, the strongly connected parent resembles to the physical parents of the hierarchical database models such as IMS. In this case, the logical parents is realized through our referencing edges.

For the rest of the paper, we shall refer strongly connected ancestor simply as ancestor. Likewise, we shall refer strongly connected descendent, strongly connected parent and strongly connected child as descendent, parent and child respectively.

2.3 Anomalies of Semi-Structured Data

A database in a RDBMS can be considered as a special form of semi-structured data. In general, semi-structured data involves anomalies which are similar to those identified for 1NF relations. Before we illustrate the various anomalies, we will first define the database instance – a semi-structured data graph, which is an image of a given schema.

Definition 3. A *semi-structured data graph* D with respect to a S3-Graph G is a graph showing a database instance such that

1. (Node correspondence)

Each node v in D is associated with one and only one node V of role R in G . We call V in the schema the *definition node* of the data node v . Furthermore, v is playing the role R .

2. (Edge correspondence)

A component edge in D is represented by a solid arrow line. If v_1 is connected to v_2 via a component edge e with tag T , then their definition nodes in G must be likewise connected via a component edge E with tag T , with or without the suffix “*”. We call the edge E in G the *definition edge* of the

edge e in D . Likewise for the referencing edge which is represented by a dashed arrow line.

3. (Root correspondence)

For each node v , there must be another node w whose definition node is a root node in G , such that either v is w itself or v is connected to w via component edges.

4. (Data type correspondence)

Each node v whose definition node is a leaf entity node V with data type $Type$ in G must be associated with a data of the matching data type $Type$.

5. (Cardinal correspondence)

For each component edge e with tag T connecting u to v , if the tag associated with e 's definition edge is not suffixed with "*", then u cannot connect to another node w with the same tag T .

Furthermore, v_1 in D is an ancestor of v_2 in D if the definition node of v_1 is an ancestor of the definition node of v_2 . The definition of descendant, parent and child can be defined similarly.

Finally, we say that the semi-structured data graph D is an *image* of G .

Figure 1 is a semi-structured data graph which is an image of the S3-Graph in Figure 2. This design is not a good design because many anomalies occurs. For example, each tutor has one name, but this information may appear many times in a semi-structured data graph as each tutor may teach many students. For example, the name "Tan" is repeated twice in the semi-structured data graph in Figure 1. Now, we encounter some **anomalies**. If a female tutor needs to change her surname after marriage, we must make sure that all the appearances of this information are consistent. This is the **rewriting anomaly**. Similarly, if the course is not mounted, the information about the tutor may be deleted together with its parent. This is the **deletion anomaly**.

Note that due to the flexibility of semi-structured data, there is no **insertion anomaly**. To insert a tutor and his name, it is possible to insert a new tree that indicates the above information.

In a semi-structured data graph, an object instance can be connected to multiple occurrences of objects of the same role. This introduces other types of anomalies that do not happen in relation in RDBMS that permits only atomic-valued attributes. One of the anomalies can be illustrated as follows.

Refer to Figure 1 again, a tutor can have a set of offices. Since a tutor may appear more than once in the graph, the information "Tan has an office at S16 05-13", which is *independent* of the Student and Course the tutor teaches, is repeated twice in the graph. This introduces anomaly. We refer to such anomaly as **set anomaly**.

3 A Normal Form for Semi-Structured Schema Graph (S3-NF)

In order to remove the anomalies that may exist in a given semi-structured data, we define a normal form for it. In this section, the concept of SS-dependency and this new normal form, called S3-NF, is described.

3.1 SS-Dependency

Definition 4. Given a S3-Graph G , and let $A = \ll A_1, \dots, A_m \gg$ be a sequence of roles in G . The sequence A is called *hierarchical role sequence* if A_i is an ancestor of A_j whenever $j > i$.

Definition 5. Given a S3-graph G , and a semi-structured data graph D which is an image of G . Let $A = \ll A_1, \dots, A_m \gg$ be an hierarchical role sequence in G such that A_i is an ancestor of A_j whenever $j > i$. An *instance* of A wrt G in D is a sequence of nodes, $\ll a_1, \dots, a_m \gg$ in D such that a_i is of role A_i and a_i is the ancestor of a_j whenever $j > i$.

Definition 6. Given a S3-graph G , and a semi-structured data graph D which is an image of G , let $e = \ll a_1, \dots, a_m \gg$ and $e' = \ll a'_1, \dots, a'_m \gg$ be two instances in D of an hierarchical role sequence $A = \ll A_1, \dots, A_m \gg$ in G , e agrees e' if and only if for every corresponding node a_i in e and a'_i in e' , we have

1. If a_i and a'_i are atomic data, then they have the same value.
2. If a_i and a'_i are objects, then they represent the same object.¹
3. If a_i and a'_i are references to another objects, then the two object instances referenced by a_i and a'_i are the same object.

Example 3. Refer to Figure 3, the instance $\ll \&8' \gg$ agrees with the instance $\ll \&8'' \gg$. as both of them reference the same object represented in $\&8$. On the other hand, the instance $\ll \&3, \&5 \gg$ does not agree with the instance $\ll \&4, \&10 \gg$ as their course codes are different.

Definition 7. With respect to a S3-Graph G , for a hierarchical role sequence $\ll A_1, \dots, A_m \gg$, where A_i is an ancestor of A_j whenever $i < j$, and a single entity type B where B is a descendent of A_m , we have $A_1, \dots, A_m$ SS-determines B , denoted as

$$A_1, \dots, A_m \implies B$$

if in any semi-structured data graph D which is an image of G , whenever any two different instances e_1 and e_2 of A in D agree, it implies that the set of instances in D of role B having e_1 as an ancestor is the same as that of having e_2 as an ancestor.

¹ Deciding if two objects are the same depends on the underlying database model. In general, it can be decided by at least two ways: same key value, and same object-id.

Example 4. Refer to the S3-Graph in Figure 2, we have $Tutor \Rightarrow Office$. The set of offices that a tutor has solely dependent on the tutor, and is not dependent on the courses he/she teaches, nor the students he/she has. This SS-dependency can be illustrated by one of its images as shown in Figure 1: The two instances “<< 8 >>” and “<< 13 >>” agree as they represent the same tutor “Tan”. Their office also agree as they both represents the same object “S16 05-13”.

On the other hand, $Tutor \not\Rightarrow Feedback$. Refer to Figure 1, although the instance “<< 8 >>” agrees with “<< 13 >>”, their descendent instances of role Feedback, “<< 16 >>” and “<< 23 >>”, do not agree. For this database, the correct SS-dependency should be $Student, Course, Tutor \Rightarrow Feedback$.

Theorem 1. *Let G be a S3-Graph, A and B be two hierarchical role sequences such that any of the roles of A is an ancestor of each role in B , We have the following properties for SS-dependency:*

1. (reflexivity) For any A , $A \Rightarrow A$.
2. (generalization of functional dependency) if $A \rightarrow B$, then $A \Rightarrow B$.
3. (left augmentation) Let C be a role that is an ancestor of each role in B and $A \Rightarrow B$, then $AC \Rightarrow B$ where AC represents the hierarchical role sequence containing all the roles in A and C ,
4. (right augmentation) Let C be a role that is a descendent of each role in B and $A \Rightarrow B$, then $A \Rightarrow BC$. where BC represents the hierarchical role sequence containing all the roles in B and C ,
5. (transitivity) For any three hierarchical role sequences A, B, C , if $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$.

Proof: The proof of these properties follows directly from the definitions of SS-dependency and functional dependency.

Definition 8. Let A and B be two hierarchical role sequences, If there exists a hierarchical role sequence C such that

$$\begin{aligned} A &\Rightarrow B \text{ and} \\ B &\Rightarrow C \text{ and} \\ B &\not\rightarrow A \end{aligned}$$

then we say that C is *transitively SS-dependent* on A via B .

Example 5. Refer to Figure 2, Code, are transitively SS-dependent on Student via Course since

$$\begin{aligned} Student &\Rightarrow Course \\ Course &\Rightarrow Code \\ Course &\not\rightarrow Student \end{aligned}$$

Theorem 2. *Given a S3-Graph G , if a role C in G is transitively SS-dependent on another role A via role B , then there exists a semi-structured data graph D which is an image of G such that the rewriting anomaly occurs upon updating the data of role C .*

Proof: We can build a semi-structured data graph as follows,

1. We first construct two instances a_1 and a_2 of role A such that they do not agree.
2. we construct the descendents of a_1 and a_2 such that they are of role B and represent an identical object. Since $B \not\rightarrow A$, it does not contradict the assumption that a_1 and a_2 does not agree.
3. Since $B \Rightarrow C$, for the set of descendents of b_1 which plays the role C , it must be the same as the set of descendents of b_2 with the role C .

We have now constructed a semi-structured data graph. If we update the information of C under b_1 , it can cause inconsistency unless updating is also done at the same time to the information of C under b_2 .

Hence, there is a semi-structured data graph D , which is an image of the given S3-Graph G , such that the rewriting anomaly occurs when we update C .

3.2 S3-NF and decomposition of a S3-Graph

Definition 9. A S3-Graph G is said to be in S3-NF if there is no transitive SS-dependency in the graph.

In order to restructure a S3-graph to reduce redundancy, we need to remove any transitive SS-dependency in a given S3-Graph. If this can be done, then the schema will be in S3-NF. In this paper, we adopt the decomposition approach to remove transitive dependencies. However, as in the case of relational database, decomposition approach by no means ensures a good solution. Integrity constraint information can be lost during the decomposition. Indeed, as mentioned in [12], it is not always possible to remove every transitive dependency in a nested relation solely by decomposition. As semi-structured data is even more flexible than nested relation, our decomposition method can only transform the schema to reduce redundancy, but may not always remove all transitive dependencies and achieve S3-NF. In future research, we will propose another synthesis method similar to Bernstein [2] and Ling's [11] method to generate a S3-NF scheme and at the same time, guarantee that no constraint information is lost.

The basic operation of our restructuring is to introduce new reference nodes and decompose the given schema graph. The main goal is to remove transitive SS-dependency in the graph. This can be done by the following step:

Given a S3-Graph G , we can decompose it to reduce redundancy.

1. For each role B , if there does not exist a role set A such that
 - (a) $A \Rightarrow B$ and
 - (b) $B \not\rightarrow A$,
 skip the rest of the following steps and continue to check for another role.
2. Let C_j be the set of the children of B such that
 - (a) $B \Rightarrow C_j$,
 - (b) for every descendent of C_j which is of role D , we have $B \Rightarrow D$.
 If there is no such C_j , then skip the rest of the following steps and continue to check for another entity type.

3. (Graph decomposition) Otherwise, duplicate the node V which has the role B to form a new node V' . It will be the root of a new tree. Move each of the C_j and all the descendents of C_j and their corresponding edges under V' . Now, replace the original node V by a reference node. This reference node shall reference V' . The tag of the referencing edge will be B .

Example 6. Refer to the schema represented in the S3-Graph in Figure 2, we want to restructure the schema to reduce redundancies.

1. We first inspect the role *Student*. Since there is no other role A such that $A \Rightarrow \textit{Student}$, no redundancy will be caused by the *Student* role.
2. The next role is *Name*. We have $\textit{Student} \Rightarrow \textit{Name}$, but *Name* has no descendent. Our algorithm skips this role and checks for other roles.
3. For the role *Course*, we have $\textit{Student} \Rightarrow \textit{Course}$. Consider the node #3 in Figure 2, it has four children: #4, #5, #6 and #7. For #4 which represents *Code*, since $\textit{Course} \Rightarrow \textit{Code}$, hence, *Code* is one of the C_j . Similarly, *Title* is another C_j as $\textit{Course} \Rightarrow \textit>Title$. Note that *Grade* and *Tutor* are not in C_j as $\textit{Course} \not\Rightarrow \textit{Grade}$ and $\textit{Course} \not\Rightarrow \textit{Tutor}$. We now decompose this graph. The subtree *Course*, *Code* and *Title* are disconnected from *Student*. The original node #3 is renamed as #3'. It now becomes a reference node which references the entity node *Course* (node #3).
4. Similarly, when we inspect the role *Tutor*, we find out that
 - (a) We have $\textit{Student}, \textit{Course} \Rightarrow \textit{Tutor}$, but $\textit{Tutor} \not\Rightarrow \textit{Student}, \textit{Course}$.
 - (b) As $\textit{Tutor} \Rightarrow \textit{Tutor.Name}$, *Tutor.Name* is one of the C_j which can cause anomaly.
 - (c) *Office* is also in C_j as firstly, $\textit{Tutor} \Rightarrow \textit{Office}$, Furthermore, for the two children of *Office*: *Building* and *Room*, our algorithm needs to verify that the set of *Room* and *Building* that a tutor has can be solely dependent on *Tutor* only, and does not require extra information such as *Course* and *Student*.
 - (d) Finally, *Feedback* is not one of the C_j since $\textit{Tutor} \not\Rightarrow \textit{Feedback}$.
 Another decomposition is done to duplicate node #7.
5. No other remaining role requires further decomposition. Our restructuring step stops.

The final restructured S3-Graph is shown in Figure 4. It is also in S3-NF.

4 ER Approach to Semi-Structured Database Design

The task of designing “good” semi-structured database can be made easier if we have more semantics. [9] proposed a normal form for the ER model. Using this ER normal form, [10] can give a good design for nested relations. Using this idea, we can also make use of the ER normal form to design good semi-structured database. This top-down approach [10], which consists of normalizing an ER diagram and converting a normalized ER diagram into the semi-structured database has two advantages:

1. Normalizing an ER diagram effectively removes ambiguities, anomalies and redundancies at a semantic level.
2. Converting a normalized ER diagram into semi-structured database results in a database schema with clean semantics.

In this section, we will discuss briefly how we can use the ER approach to design good semi-structured databases.

The ER approach uses the concepts of entity types and relationship sets to capture real world semantics. An entity type or relationship set has attributes which represents its structural properties. Attributes can be single-valued or multivalued. Figure 5 shows the ER diagram for our Student-Course-Tutor example. We see that students, courses and tutors are modeled as entity types Student, Course and Tutor respectively. Student has an attribute Name while Course has attributes Code and Title. Tutor has a single-valued attribute Name and a composite multivalued attribute Office. Here, we assume that Student.Name, Course.Code and Tutor.Name are the identifiers of the entity types Student, Course and Tutor respectively. The relationship set Enrol captures the association that a student is enrolled in a course and has a single-valued attribute Grade. Since a student taking a course is taught by some tutors, we need to associate the relationship set Enrol with the entity type Tutor. This is accomplished using aggregation which effectively allows us to view Enrol as an entity type for the purpose of participation in other relationship sets. This association is captured in the relationship set SCT. Feedback is a single-valued attribute in SCT as its value is given by the student for each tutor teaching him in some course. It is clear that the ER diagram is also in normal form [9].

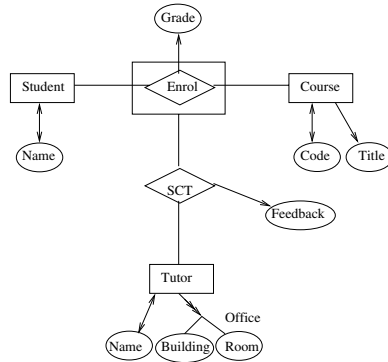


Fig. 5. Entity-Relationship Diagram for Student-Course-Tutor Example

We now outline the translation of the normal form ER diagram into a S3-graph. Details of the translation algorithm will be given in a full paper.

1. Each entity type E becomes an entity node N with role E . Each attribute A of E is a node which is connected to E by a component edge with tag A .

2. For each n -ary relationship set R , we first construct a path to link the participating entity types of R . Let $\prec V_1, V_2, \dots, V_k \succ$ be the path. Vertex V_1 corresponds to some participating entity type of R which is associated with some entity node N_1 . Each vertex V_i , where $2 \leq i \leq n$, corresponds to either a participating entity type of R or a combination of two or more participating entity types of R . We next create reference nodes $N_2, \dots, N_k$ that is associated with $V_2, \dots, V_k$ respectively. Then we have a component edge from N_i to N_{i+1} , where $1 \leq i \leq k - 1$. Each reference node N_i , where $2 \leq i \leq k$, also has referencing edge(s) to the entity node(s) that is associated with the participating entity type(s) of R corresponding to V_i . Any attribute A of R is a node which is connected to N_k by a component edge with tag A . Note that relationships which are involved in aggregations have to be processed first because they will establish portions of a subsequent path.

The ER diagram in Figure 5 can be translated to the semi-structured schema graph in Figure 4 as follows. The entity types Student, Course and Tutor become entity nodes #1, #3, #7 respectively. The attributes also become nodes and are connected to their owner entity type by component edges. We need to process the relationship Enrol before SCT because Enrol is involved in an aggregation. Suppose we choose to construct the path $\prec Student, Course \succ$ from the participating entity types of Enrol, then the relationship set Enrol becomes a reference node #3', and the entity node #1 has a component edge to #3', which in turn has a referencing edge to the entity type #3. The attribute Grade is a component of the reference node #3'. The relationship set SCT also becomes a reference node #7'. The path corresponding to SCT must be $\prec Student, Course, Tutor \succ$ because Enrol is an aggregate in the relationship set SCT and it has earlier established the $\prec Student, Course \succ$ portion of the path. Node #3' has a component edge to #7' which in turn has a referencing edge to #7. The attribute Feedback is a component of the reference node #7'. We observe that the semi-structured schema graph obtained is not unique but is dependent on the path constructed.

5 Conclusion

In this paper, we have shown the importance of designing good semi-structured databases. We defined a semi-structured schema graph called S3-graph for semi-structured databases. We identified various anomalies, which includes rewriting anomaly, deletion anomaly and set anomaly, that may arise if a given semi-structured database is not designed properly and contains redundancies. We proposed a normal form for semi-structured schema graph, S3-NF. We presented two approaches to design good semi-structured databases, namely, the restructuring approach and the ER approach. The former uses the decomposition technique to normalize a semi-structured schema graph which may not guarantee a good solution while the latter uses the normal form ER model to obtain a normal form semi-structured schema graph. Our definition of the semi-structured schema graph attempts to correspond to the XML definition so that we can apply our technique to design good XML databases in future.

References

1. S. Abiteboul, D. Quass, J. Widom, and J. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1), 1997.
2. P.A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems*, 4(1):277–298, 1976.
3. T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (xml) 1.0. W3C Recommendation available at <http://www.w3.org/TR/1998>, 1998.
4. P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to semistructured data. In *Int. Conference on Database Theory*, 1997.
5. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. ACM SIGMOD*, 1996.
6. E.F. Codd. Further normalization of the database relational model. Database Systems, edited by Randell Rustin, 1972.
7. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3), 1997.
8. R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proc. of the 23rd VLDB*, 1997.
9. T.W. Ling. A normal form for entity-relationship diagrams. In *Proc. of 4th Int. Conference on Entity-Relationship Approach*, pages 24–35, 1985.
10. T.W. Ling. A normal form for sets of not-necessarily normalized relations. In *Proc. of 22nd Hawaii Int. Conference on Systems Science*, pages 578–586, 1989.
11. T.W. Ling, F.W. Tompa, and T. Kameda. An improved third normal form for relational databases. *ACM Transactions on Database Systems*, 2(6):329–346, 1981.
12. T.W. Ling and L.L. Yan. Nf-nr: A practical normal form for nested relations. *Journal of Systems Integration*, 4:309–340, 1994.
13. D. Maier. Theory of relational databases. Pitman, 1983.
14. A. Makinouchi. A consideration on normal form of not-necessarily normalized relation in the relational data model. In *Proc. of 3rd VLDB*, 1977.
15. J. McHugh, S. Abiteboul, R. Goldman, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), 1997.
16. S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Objects: Concise representation of semistructured hierarchical data. In *Proc. of the 13th Int. Conference on Data Engineering*, 1997.
17. Z.M. Ozsoyoglu and L.Y. Yuan. A normal form for nested relations. *ACM Transactions on Database Systems*, 1(12):111–136, 1987.
18. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, pages 251–260, 1995.
19. M.A. Roth and H.F. Korth. The design of 1nf relational databases into nested normal form. In *Proc. of ACM SIGMOD*, 1987.
20. J.D. Ullman. Principles of database systems. Computer Science Press, 1983.
21. K. Wang and H.Q. Liu. Schema discovery from semistructured data. In *Int. Conference on Knowledge Discovery and Data Mining*, 1997.

Building Views over Semistructured Data Sources^{*}

Silvana Castano¹ and Valeria De Antonellis²

¹ University of Milano, DSI - via Comelico, 39 - 20135, Milano - Italy
castano@dsi.unimi.it

² University of Brescia, DEA - via Branze, 38 - 25123 Brescia - Italy
deantone@ing.unibs.it
deantone@elet.polimi.it

Abstract. The goal of this paper is to present concepts and techniques to build views and organize the information search space of heterogeneous semistructured data sources with respect to expected queries. We formalize the notion of object pattern and of semantic correspondence between object patterns. Object pattern analysis is ontology-driven and leads to the construction of reconciled views of the sources, called *global object patterns*. Global object patterns mediate between heterogeneous terminology and structure of data in different sources and are used for query formulation and data extraction.

1 Introduction

Research on semistructured data has recently gained attention, due to the large number of information sources available over the Web. The focus is on the development of powerful query languages facilities, to pose semantically rich queries against one or more sources. To access information over the Web, one can rely on search engines supporting a keyword-based search of documents. However, such tools lead to the retrieval of a large number of documents, often not relevant. To make possible querying a semistructured data source using a DBMS-like query language, semistructured data models have been proposed, based on the notion of graph to represent information on the structure of data in the source [1,4]. Following this approach, structured query languages have then been proposed for querying the Web, which extend the SQL syntax to the characteristics of the semistructured data model [12,10]. When dealing with restricted domains, such as Intranets, additional problems arise in querying/browsing distributed information sources, regarding how to handle semantic heterogeneity of data in different sources. In fact, in such domains, applications span many organizations, data sources of different organizations deal with related and often overlapping data, and the need arises for the involved organizations to cooperate and share data each other, despite of the fact that different vocabularies and/or different

^{*} This research has been partially funded by the *Metodologie e Tecnologie per la Gestione di Dati e Processi su Reti Internet e Intranet* - MURST ex-40% project.

data structures and representations are used in the domain. Consequently, techniques to address vocabulary and semantic heterogeneity between different data sources are required, to extend query language facilities. This way, the information search space can be organized at a global level for the domain, to facilitate the query process.

Integration and translation tools for heterogeneous data sources have been developed, and information integration architectures based on mediator/middleware and ontologies have been proposed [3,5,9,11,10,15]. We addressed schema analysis problems for structured databases in previous research work, for semantic heterogeneity and integration in [7]. Starting from these models and languages, we now study the problem of interoperating across several semistructured data sources in a given domain. The goal is to organize the corresponding information search space with respect to expected queries. We formalize the notion of object pattern and of semantic correspondences between object patterns. We perform an ontology-driven analysis of object patterns to construct reconciled views of the sources, called *global object patterns*. In this way, global object patterns mediate between heterogeneous terminology/structure of data in the different sources to support query formulation and data extraction.

The paper is organized as follows. In Section 2, we give basic definitions and the overview of our approach. In Section 3, we discuss object patterns and their extraction. In Section 4, we introduce semantic correspondences for object patterns. In Section 5, we illustrate the construction of global object patterns. In Section 6, we show the use of global object patterns for searching the information space, and, finally, in Section 7, we give our concluding remarks.

2 Basic Definitions and Overview of the Approach

According to data models that have been proposed in the recent literature [4,1], a semistructured data source is represented as a rooted, labeled tree. In the following, we formalize the definition of semistructured object and source.

Definition 1 (Semistructured object). *A semistructured object so in a source S is a triple of the form $(id_{so}, l_{so}, v_{so})$ where:*

- id_{so} is the object identifier. It univocally identifies so within the source;
- l_{so} is the object label. It denotes the “meaning” (or semantic content) of so;
- v_{so} is the object value. It is a set of pairs $(l_i : v_i)$, $i = 1, \dots, k$, where l_i is a label denoting an attribute for so and v_i can be:
 - an elementary value of a basic type, such as **string**, or
 - an object identifier, that is, a reference value to an object.

$(8, \text{WORKER}, \{(\text{Name}: \text{'Dilbert'}), (\text{Salary}: \text{'53000'}), (\text{Department}: \text{'Z11'})\})$ is an example of semistructured object describing worker information. The object is characterized by three attributes each one having an elementary value.

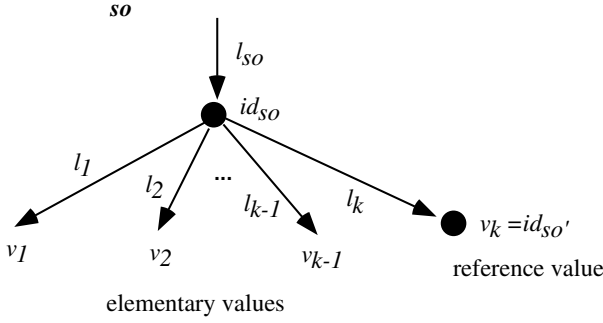


Fig. 1. Graph representation of a semistructured object so

Following recent models proposed in the literature for semistructured data, a semistructured object $so = (id_{so}, l_{so}, v_{so})$ can be graphically represented as in Fig. 1. In the paper, for the sake of clarity, object identifiers are omitted in the graphical notation.

Definition 2 (Semistructured source). A semistructured source S is a 4-tuple of the form (V, E, r, f) where:

- V is a set of vertices;
- $E \subseteq V \times V$ is a set of edges;
- $r \in V$ is the root;
- $f : E \rightarrow L_S$ is the edge labeling function; $L_S = \{l_1, l_2, \dots, l_q\}$ is a set of object/attribute names.

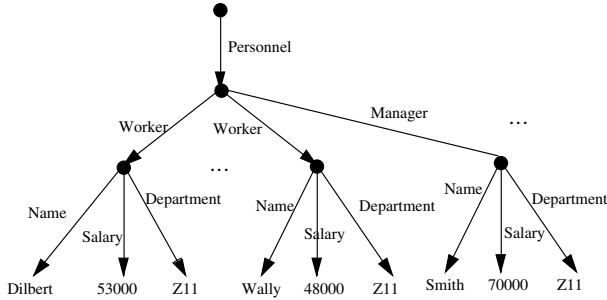


Fig. 2. Graph representation of the source S_1

An example of semistructured “Employee” source (S_1) is shown in Fig. 2, storing information about name, salary, and department of workers and managers of a company. In Fig. 3, we show a second example of “Employee” source (S_2), providing information on the staff of a certain organization. In this source, each person of the staff is characterized by information on name, weekly-pay, and

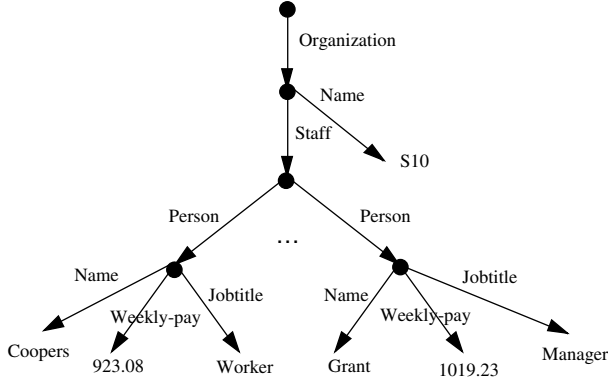


Fig. 3. Graph representation of the source S_2

jobtitle. Heterogeneities can occur in the sources: for example, information on qualification is represented as an attribute (i.e., *Jobtitle*) in S_2 and as different objects in S_1 (i.e., *WORKER* and *MANAGER* objects).

2.1 Structuring the Information Search Space by Global Object Patterns

To integrate data from different semistructured data sources, different approaches are possible. The first approach is to rely on the view definition capabilities of the query language. For example, in [8], a query language with view definition primitives is presented for XML sources. The integration is realized according to an “extemporary approach”: the user knows the location of the sources in advance and requests the data of interest contained in the sources by formulating an extraction query on the involved objects of each source. The knowledge required to the user with this approach is significant, since the user is responsible each time for discovering semantically related information for the integration, based on the meaning of the labels in the sources. The approach makes it possible to integrate data extracted from any source, provided that the user knows in advance source location and semantic content.

In order to provide a more structured information space and allow the user to pose queries against the different sources without knowing in advance the location, vocabulary, and contents of the sources, we propose a “structured” approach. With this approach, we partition the information space into properly constructed reconciled views (that we call *global object patterns*) providing the user with distinct subject categories for organizing and indexing the contents of multiple heterogeneous sources. The sources can be queried using a structured query language with the following advantages:

1. the user has to formulate only one query on the global object pattern(s) of interest;

2. the user does not have to be aware of the location, vocabulary, and contents of each source, since he operates on reference, reconciled representations of the sources;
3. the user does not have to specify how to reconstruct the integrated view of the data extracted from different sources for a given query, since the integrated result is automatically constructed by the system, by exploiting the set of references and semantic correspondences associated with the global object patterns on which the query has been formulated.

The approach for structuring the information search space in a given domain is based on the following concepts and techniques.

Object patterns. Object patterns represent the structure of data in a semistructured source. We formalize the concept of object pattern in Section 3.

Semantic correspondences between object patterns. Semantic correspondences are recognized between object patterns of different sources that describe the same concept. Semantic correspondences between object patterns are established by exploiting the knowledge provided by a domain ontology. We define semantic correspondences and techniques for discovery them in Section 4.

Global object patterns. Global object patterns are reconciled views of groups of object patterns with a semantic correspondence in different sources. Global object patterns cover the source search space for expected queries by providing a reconciled description and a set of references to semantically related object patterns in the analyzed sources. We define global object patterns and techniques for their construction in Section 5.

3 Object Patterns of a Semistructured Source

In order to reason about the contents of a semistructured source, it is useful to associate with the source a concise and convenient summary of its contents, also called schema or structure in the literature [4]. For example, when dealing with an XML source, the so called “Document Type Definition” (DTD) [8] is generally available, which specifies the allowed structure for the objects (i.e., XML documents) in the source. The issue of adding structure to semistructured data sources in general has been recently investigated in the literature, specially for browsing and for query optimization purposes [14]. In the following, we introduce the concept of *object pattern* to formally reason about the structure of a semistructured source, for structuring the information search space. Object patterns provide a concise description of all objects denoting the same concept in the source, which is representative of all their, possibly heterogeneous, representations.

Let S be a semistructured source and $G_k = \{so_1, so_2, \dots, so_p\}$ be a set of semistructured objects denoting different instances of a given concept in the source, characterized by the same label l_{so} in S .

Definition 3 (Object pattern). An object pattern op_k for a set of objects G_k of a semistructured source S is a pair of the form $op = (l_k, A_k)$, where:

- l_k is the label of the object pattern, that is, the label l_{so} common to the objects belonging to G_k ;
- $A = \{l_i \mid \exists so \in G_k, l_i \in v_{so}\}$ is the set of attribute labels defined for the objects $so_i \in G_k, i = 1, \dots, p$.

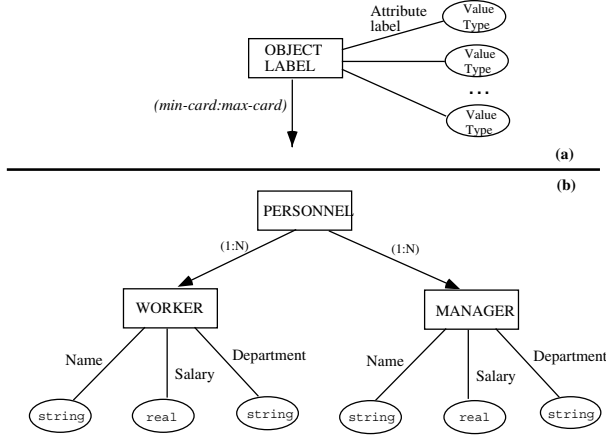


Fig. 4. Graphical representation of the structure of the source S_1 .

An object pattern is defined for each different concept in the source, by considering the set of objects describing it. Object patterns for the source S_1 are the following:

Personnel-pattern = (Personnel, { Worker+, Manager+ })

Worker-pattern = (Worker, { Name, Salary, Department })

Manager-pattern = (Manager, { Name, Salary, Department })

where multiple occurrence of a label l_i in the value set v_{so} of an object so is denoted in the object pattern by the symbol '+'. The structure of a semistructured data source is given by the union of all the object patterns defined for the objects therein contained. We adopt a graph representation for object patterns, as shown in Fig. 4(a). By composing the graphical representations of all object patterns of a given source, we obtain the structure of the source. For example, the structure of the source S_1 is shown in Fig. 4(b). We distinguish two kinds of nodes: labeled nodes denoting objects, represented by boxes, and labeled nodes denoting value types, represented by ovals. Attribute labels are associated with the edges entering in the corresponding node. When connecting objects, edges denote object references, and have an associated pair of values expressing cardinality constraints (i.e., minimum and maximum) on the object instances in the source.

4 Semantic Correspondences between Object Patterns

Semantic correspondences are established between object patterns of different, heterogeneous sources to capture the fact that the sources contain semantically related data. Semantic correspondences are discovered by comparing object patterns on the basis of knowledge provided by a reference ontology O .

4.1 Ontology Structure and Use

Ontologies have been developed to facilitate communication and agreement among multiple distributed sources/agents and to provide shared knowledge that covers a whole domain or a more specific application. The role of ontologies is to organize the knowledge about terms at a higher abstraction level into concepts and relationships among concepts. An example of Web-accessible ontology is **WordNet**, where English nouns are organized according to given terminological relationships [13]. Conventional terminological relationships generally maintained in ontologies are *synonymy*, that we denote by SYN and *hypernymy*, that we denote by BT (Broader Terms).

We conceptualize an ontology O as a graph, where nodes correspond to terms and labeled edges between nodes to relationships between terms. Terms are connected through paths involving a number of (possibly different) relationships. In the following, we denote an ontology relationship as $\mathfrak{R}$.

The ontology is exploited for assessing the level of semantic relationship between labels of object patterns, called *label affinity*. For the evaluation of label affinity, we implement an associative network on top of the ontology. Each terminological relationship $\mathfrak{R}$ is properly strengthened with a strength $\sigma_{\mathfrak{R}} \in (0, 1]$ to capture its implication for affinity, with $\sigma_{SYN} > \sigma_{BT/NT}$. The existence of at least one path in the associative network is the condition for two labels to have affinity. Label affinity is evaluated by means of an *affinity function* $A()$. Given two labels l and l' , $A(l, l')$ first verifies the existence of at least one path in O , and then computes the strength of the path as the product of all the strengths $\sigma_{\mathfrak{R}}$ involved in it. We denote by $\sigma_{\rightarrow m}$ the strength of a path of length m . If more than one path exists between l and l' , the one with the highest value of $\sigma_{\rightarrow m}$ is selected as the value of $A(l, l')$. The longer the path, the lower the value of $\sigma_{\rightarrow m}$. For a given path length, the greater the strength of the involved relationships, the higher the value of $\sigma_{\rightarrow m}$. For labels that are not related by a path or for which an entry does not exist in the ontology, $A()$ returns a null value.

Two labels l and l' have affinity, denoted by $l \sim l'$, if their affinity value $A(l, l')$ is greater than or equal to a given threshold α , that is, $l \sim l' \leftrightarrow A(l, l') \geq \alpha$.

The relevance and applicability of ontology relationships have been discussed in [5], where experimental results motivating our choices are also presented. We have developed automated procedures for ontology-based affinity evaluation in the framework of the ARTEMIS a tool environment [6].

4.2 Semantic Correspondences

Let $OP = \{op_1, op_2, \dots, op_K\}$ be the set of object patterns of all semistructured sources to be integrated. Two kinds of correspondences can be established between objects patterns, namely weak and strong.

Definition 4 (Weak semantic correspondence). Given two object patterns $op_1 = (l_1, A_1)$ and $op_2 = (l_2, A_2)$, with $op_1, op_2 \in OP$, a weak semantic correspondence, denoted by $\leftrightarrow^*$, is established between them if their labels have affinity, that is, if $l_1 \sim l_2$.

Definition 5 (Strong semantic correspondence). Given two object patterns $op_1 = (l_1, A_1)$ and $op_2 = (l_2, A_2)$, with $op_1, op_2 \in OP$, a strong semantic correspondence, denoted by $\leftrightarrow$, is established between them if both the following conditions hold:

1. $l_1 \sim l_2$;
2. $\exists (l, l'), l \in A_1, l' \in A_2, l \sim l'$

According to the above two definitions, two object patterns have a weak semantic correspondence if their labels have affinity in the ontology. Two object patterns have a strong semantic correspondence if they have label affinity and if at least one pair of their attributes has label affinity in turn. It is possible that a certain object pattern (respectively, attribute) in a source has a semantic correspondence with more than one object pattern (respectively, attribute) in another source. All semantic correspondences are considered in this case.

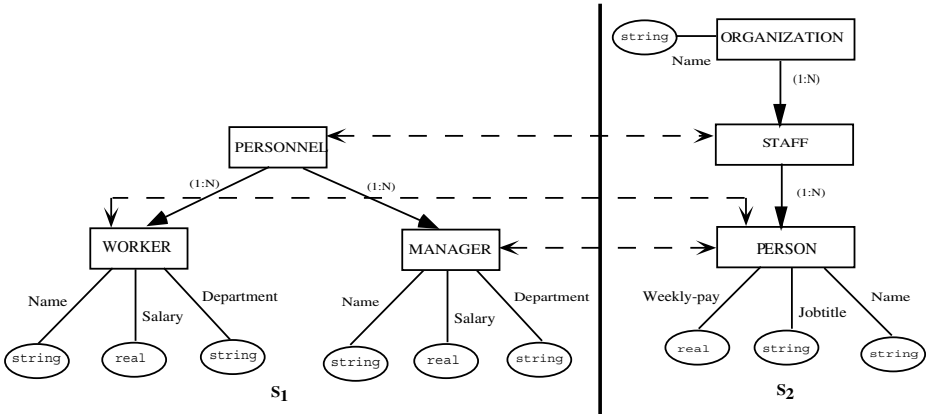


Fig. 5. Semantic correspondences between object patterns of S_1 and S_2

Example 1. A strong semantic correspondence is established among object patterns **PERSONNEL** of S_1 and **STAFF** in S_2 , as shown in Fig. 5 using a dashed line. In fact, their labels have affinity since **Personnel** SYN **Staff** in WordNet and their attribute labels have affinity in turn, since **Worker** SYN **Person** in WordNet. Another strong semantic correspondence is established between object pattern **WORKER** in S_1 and object pattern **PERSON** in S_2 . $\square$

Based on $\leftrightarrow$ and $\leftrightarrow^*$, clusters of object patterns in OP are defined. For any object pattern $op_l \in OP$, the cluster for op_l , denoted by $[op_l]$, contains:

- all object patterns having a semantic correspondence $\leftrightarrow^*$ with op_l , and
- all object patterns having a semantic correspondence $\leftrightarrow$ with op_l ,

that is, $[op_l] = \{op_k \in OP \mid op_k \leftrightarrow^* op_l\} \cup \{op_k \in OP \mid op_k \leftrightarrow op_l\}$. We denote by $OP^E = \{[op_l] \mid op_l \in OP\}$ the set of clusters resulting from OP . Among clusters, we are interested in the so-called “well-formed” clusters, denoted by $[]^{wf}$. $[op_l]^{wf}$ is a cluster that contains (at least) one object pattern (or attribute) from each source.

Example 2. The following well-formed clusters are formed for object patterns of sources S_1 and S_2 of our example:

$$\begin{aligned} [S_1.WORKER]^{wf} &= \{S_1.WORKER, S_1.MANAGER, S_2.PERSON\} \\ [S_2.PERSONNEL]^{wf} &= \{S_1.PERSONNEL, S_2.STAFF\} \end{aligned}$$

$\square$

5 Global Object Patterns

Global object patterns are reconciled views of a given cluster. Global object patterns are exploited by the user to browse and query the underlying data sources in a uniform way, based on the reference terminology and structure defined for them.

Definition 6 (Global Object Pattern). Let $[op_l]$ be a cluster of object patterns. The global object pattern obtained from the unification of the object patterns in $[op_l]$ is defined as a triple $gop_{[op_l]} = \langle l_{[op_l]}, A_{[op_l]}, [op_l] \rangle$ where:

- $l_{[op_l]}$ is the label of the global object pattern;
- $A_{[op_l]}$ is the set of attributes of the global object pattern, obtained from the unification of attributes of the object patterns in $[op_l]$.
- $[op_l]$ is the cluster from which $gop_{[op_l]}$ is derived.

A global object pattern $gop_{[op_l]}$ is defined for each cluster $[op_l]$ and is representative of all object patterns contained in $[op_l]$, since it reconciles their labels and structure. The fact that a cluster is well-formed has an impact on the kind of reconciled view provided by the corresponding global object pattern on the underlying data sources. In presence of a well-formed cluster, the global object

pattern obtained from it is marked as *featuring*, to express the fact that it provides a full view of the sources. Full view means that an object pattern can be found in each source for answering a query posed against the global object pattern. Global object patterns that are not featuring provide a “partial view” of the sources. Partial view means that an object pattern can be found only in some sources for answering a query posed against a global object pattern. Let us now describe how the steps for the construction of global object patterns.

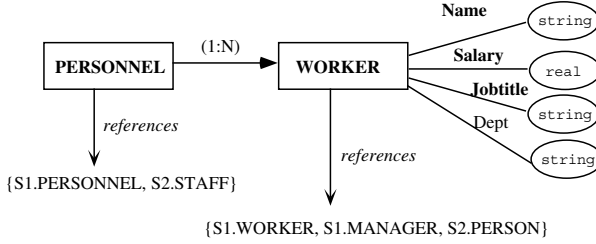


Fig. 6. Global object patterns for sources S_1 and S_2

Reconciliation of object patterns.

Labels of object patterns in $[op_l]$ are unified into the label $l_{[op_l]}$ in the global object pattern. Precisely, $l_{[op_l]}$ can coincide with one of the labels of object patterns in $[op_l]$ or it can be one of their hypernyms or synonyms, selected by exploiting the ontology relationships. We say that $l_{[op_l]}$ is representative of all object patterns belonging to $[op_l]$, and thus reconciles them. If $[op_l]$ is well-formed, $gop_{[op_l]}$ is marked as *featuring*.

Reconciliation of object pattern attributes.

Attribute labels of object patterns in $[op_l]$ are unified into a global representative label in $A_{[op_l]}$, by exploiting the ontology relationships as in the previous step. If, for an attribute in $A_{[op_l]}$, a value can be defined for each involved source, the attribute is marked as *featuring* in the global object pattern.

In order to be exploited for query purposes, a set of *references* is maintained for global object patterns, referencing the source object patterns that have been unified into the global object pattern. In particular, for each global object pattern we maintain references between: i) the global object pattern and all corresponding object patterns belonging to $[op_l]$, ii) reference reconciled attributes $A_{[op_l]}$ and the corresponding attributes they have been derived from in $[op_l]$.

Example 3. Two global object patterns PERSONNEL and WORKER are defined for sources S_1 and S_2 as shown in Fig. 6, based on the semantic correspondences between their object patterns (see Fig. 5). A relationship exists between PERSONNEL and STAFF, due to the fact that object patterns PERSONNEL of S_1 and STAFF

of S_2 have a strong semantic correspondence and that a global object pattern **WORKER** has been defined in correspondence of the object patterns referred by each of them (i.e., **WORKER** and **MANAGER** in S_1 and **PERSON** in S_2). The global object pattern derived from cluster $l_{[\text{WORKER}]}$ is called **WORKER** since worker and person are synonyms and person is a hypernym of manager in the ontology. The global attribute **Name** is defined in **WORKER** to unify attributes **Name** of S_1 .**WORKER** and S_1 .**MANAGER** and of S_2 .**PERSON**. The attribute **Salary** is defined to unify attributes **Salary** of both S_1 .**WORKER** and S_1 .**MANAGER** and attribute **Weekly_Salary** of S_2 .**PERSON**. In fact, salary is a hypernym of weekly_salary in the ontology. Both attributes are examples of featuring attributes, in that they have a value for objects for each considered source. For each global object pattern in Fig. 6, we also show its references to corresponding object patterns in sources S_1 and S_2 . $\square$

6 Searching the Information Space

In this section, we describe the role of global object patterns for evaluation of queries against the global information search space, that is, queries that span multiple data sources. We envisage two main query strategies, to support the user in finding distributed information, namely a content-based and an index-based search.

6.1 Content-Based Search

According to their definition, global object patterns allow mediation between heterogeneous terminologies/structures in different sources. For each user query, they are used to identify local object patterns for answering the query, and, at the end, to reconstruct the integrated result from the sources involved in the query. A global query $\bar{q}$ is formulated on available global object patterns by selecting attributes of interest, possibly specifying filtering conditions on them.

Example 4. Consider, for example, the following content-based query on S_1 and S_2 : *Retrieve names and salaries of managers.*

The following global query $\bar{q}_1$ is formulated by the user based on the labels and structure of the global object patterns of Fig. 6, using an SQL-like query language:

```
 $\bar{q}_1$ :      SELECT Worker.Name, Worker.Salary
           WHERE Worker.JobTitle LIKE 'Manager'
```

$\bar{q}_1$ involves the global object pattern **WORKER**, and has the **Name** and **Salary** properties in the target list, and a filtering condition on property **Job** in the **WHERE** clause. $\square$

Answering a global query $\bar{q}$ on global object patterns is performed in the following steps.

Decomposition of $\bar{q}$ into sub-queries on data sources. Query $\bar{q}$ is decomposed into sub-queries on each involved source, by exploiting references associated with global object patterns. Query decomposition consists in replacing labels of q by labels used in each source, by exploiting references maintained for global object patterns. Each sub-query is then processed against each single source, by exploiting its local object patterns, to retrieve source objects matching the query. For each local object pattern, an “inverted” structure can be maintained, to keep references to the source objects that must be used to answer the query posed against the object pattern.

Merging of sub-query answers. Answers to sub-queries are set of objects. For presentation to the user, source objects retrieved for each sub-query have to be translated according to the structure of the global object pattern(s) appearing in $\bar{q}$. Translation is performed by means of conversion functions which replace each label of retrieved objects with the corresponding label used in the global object pattern, by exploiting the references.

Example 5. The global query $\bar{q}_1$ is decomposed into the following two sub-queries for sources S_1 and S_2 , respectively:

- q_{11} (on source S_1):
`SELECT  $S_1$ .Manager.Name,  $S_1$ .Manager.Salary`
- q_{12} (on source S_2):
`SELECT S_2 .Person.Name, S_2 .Person.(Weekly-pay*4)
WHERE Person.Jobtitle LIKE ‘‘Manager’’`

□

The decomposition of a global query into sub-queries is performed according to stored “mapping and conversion functions”, defined for global object patterns to populate the pattern with data stored at each source.

Mapping functions

A mapping function for a global attribute $\bar{a}$, denoted by $\rightarrow^M(\bar{a})$, expresses how to map global query expressions involving $\bar{a}$ into corresponding query expressions involving object patterns of the cluster from which $\bar{a}$ has been derived. An example of mapping function, defined for the global property *Jobtitle* of global object pattern *WORKER* (see Fig. 5) is the following:

```

 $\rightarrow^M(\text{Jobtitle})$ 
  if Jobtitle=‘‘Manager’’ then
     $S_1$ : SELECT MANAGER
     $S_2$ : Person.Jobtitle = ‘‘Manager’’
  if Jobtitle=‘‘Worker’’ then
     $S_1$ : SELECT WORKER
     $S_2$ : Person.Jobtitle = ‘‘Worker’’

```


This mapping function specifies the query statements that have to be defined in the sub-queries on sources S_1 and S_2 to correctly retrieve the values of interest for **Jobtitle** in a global query, by taking into account the structure of object patterns in both S_1 and S_2 . In particular, the function specifies that the expression **Jobtitle**=‘**Manager**’ (respectively, **Jobtitle**=‘**Worker**’) in a query $\bar{q}$ is mapped into: i) a **SELECT MANAGER** (respectively, **SELECT WORKER**) clause in the sub-query for S_1 , since in this source we have a **MANAGER** (respectively, **WORKER**) object pattern representing the managers (respectively, workers) in the source; ii) a **PERSON.Jobtitle**=‘**Manager**’ statement in the sub-query for S_2 , since in this source qualification of employed people is represented in form of attribute in the object pattern **PERSON**.

Conversion functions

A conversion function for a global attribute $\bar{a}$ is defined to implement the necessary transformations to convert the values retrieved from each source with respect to the values expected for $\bar{a}$ at the global level [9,17]. A kind of conversion can be necessary for assigning the right value to a global attribute based on the meta-data (e.g., format, unit, currency, levels of precision) of the attributes of the object patterns in each respective local source. For example, for the **Salary** global attribute, a conversion function has to be defined since salary values in S_1 and S_2 are stored at different precision levels, that is, monthly and daily values. The conversion function for **Salary** is defined as follows:

$$\begin{aligned} \rightarrow^M(\text{Salary}) = \\ & S_1: \text{MANAGER.Salary, WORKER.Salary} \\ & S_2: \text{Person.Weekly-pay*4} \end{aligned}$$

Another kind of conversion can be necessary for assigning the right value to a global attribute for source objects for which the global attribute is not defined. For example, for the **JobTitle** global attribute, a conversion function has to be defined since an attribute expressing person qualification is not defined S_1 . The conversion function for **JobTitle** is defined as follows:

$$\begin{aligned} \rightarrow^M(\text{JobTitle}) = \\ & S_1: \quad \text{if } S_1.\text{MANAGER} \text{ then JobTitle}=\text{“Manager”}; \\ & \quad \text{if } S_1.\text{WORKER} \text{ then JobTitle}=\text{“Worker”} \\ & S_2: \quad \text{JobTitle}=\text{Person.JobTitle} \end{aligned}$$

6.2 Index-Based Search

Global object patterns are a means to correlate descriptions of semantically related semistructured objects, and therefore provide a basis also for indexing semistructured data sources. Labels of global object patterns can be used as terms of an indexing system for source object patterns, following the conventional information retrieval approach [16]. By index-based search, specifying labels of

interest, it is possible to locate all sources concerned with a particular topic. By construction, labels of global object patterns allow satisfaction of the indexing exhaustivity property, that is related to the degree to which all aspects of the considered data sources are recognized in the indexing system.

Another important property to consider, is the term specificity. It refers to the degree of breadth or narrowness of the terms. In general, the use of broad terms results in a large number of retrieved items (both relevant and non), while narrow terms retrieve fewer items (most of them are relevant). In case of global labels, the terminology is strictly related to that of the object patterns, but can be properly controlled by exploiting related terms and corresponding relationships in the underlying ontology.

7 Concluding Remarks

In the paper, we have presented a set of techniques for structuring the information search space with multiple semistructured data sources, to build global object patterns. We introduced object patterns to formalize the notion of structure of a semistructured source, for ontology-based analysis of multiple semistructured data sources. We have discussed the use of global object patterns to mediate between heterogeneous terminology and structure in different sources, to ease query formulation at the global level for the user.

The ARTEMIS tool environment supports our pattern-based approach for structuring the information search space with multiple semistructured data sources. ARTEMIS has been developed for reconciling heterogeneous databases [6], and is integrated in a conventional mediator/wrapper system, called MOMIS [3]. Now, ARTEMIS is being extended to cope with semistructured source reconciliation. For each data source, a wrapper component is responsible for extracting the object patterns from the source, which are represented into the internal object-oriented data representation language used in ARTEMIS. The *Affinity Analysis* module of ARTEMIS implements the ontology-based analysis of object patterns for the establishment of semantic correspondences. Moreover, it defines clusters and helps the construction of corresponding global object patterns, which is an interactive process. The *Mapping engine* module implements the core functionalities for mapping global object patterns onto source object patterns for query processing.

Future research work is proceeding in the following directions: i) formalization, refinement, and experimentation of the integration-based method with more complex object pattern structures, such as, for example, DTDs and XML data sources; ii) strategies for global object pattern maintenance; iii) query optimization based on global object patterns.

References

1. Abiteboul, S.: Querying Semistructured Data. In: Proc. of the Int. Conf. on Database Theory (ICDT'97), Delphi, Greece (1997). 146, 147

2. Abiteboul, S., et al.: Tools for Data Translation and Integration. *IEEE Data Engineering Bulletin*, **22**(1) (1999) 3-8
(<http://www.research.microsoft.com/research/db/debull/99mar/issue.htm>).
3. Bergamaschi, S., Castano, S., Vincini, M.: Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record*, **28**(1) 1999, (<http://www.acm.org/sigmod/record/>). 147, 159
4. Buneman, P.: Semistructured Data. In *Proc. of Symposium on Principles of Database Systems (PODS'97)*, Tucson, Arizona (1997) 117-121. 146, 147, 150
5. Castano, S., De Antonellis, V.: A Discovery-Based Approach to Database Ontology Design. *Distributed and Parallel Databases, Special Issue on Ontologies and Databases*, **7**(1) 1999. 147, 152
6. Castano, S., De Antonellis, V.: A Schema Analysis and Reconciliation Tool Environment for Heterogeneous Databases. In *IEEE Proc. of IDEAS'99 Int. Database Engineering and Applications Symposium*, Montreal (1999). 152, 159
7. Castano, S., De Antonellis, V., De Capitani Di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. To appear on *IEEE Trans. on Knowledge and Data Engineering* (1999). 147
8. Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: XML-QL: A Query Language for XML. World Wide Web Consortium, Working paper (1998), (<http://www.w3.org/TR/NOTE-xml-ql>). 149, 150
9. Haas, L.M., et al.: Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, **22**(1) (1999) 31-36, (<http://www.research.microsoft.com/research/db/debull/99mar/issue.htm>). 147, 158
10. Levy, A., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: *Proc. of the 22nd Int. Conf. on Very Large Data Bases - VLDB'96*, (1996) 251-262. 146, 147
11. Mena, E., Kashyap, V., Sheth, A., Illarramendi, A.: OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. In: *Proc. of First IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'96)*, Brussels (1996) 14-25. 147
12. Mendelzon, A., Mihaila, G., Milo, T.: Querying the World Wide Web. In: *Proc. of First Int. Conf. on Parallel and Distributed Information Systems*, (1996). 146
13. Miller, A.G.: WordNet: a Lexical Database for English. *Communications of the ACM*, **38**(11), (1995) 39-41. 152
14. Nestorov, S., Abiteboul, S., Motwani, R.: Extracting Schema from Semistructured Data. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD'98)*, (1998). 150
15. Papazoglou, M.P., Milliner, S.: Subject-based Organization of the Information Space in Multi-Database Networks. In: *Proc. of 10th Conf. On Advanced Information Systems Engineering (CaiSE'98)*, Pisa, Italy, (1998) 251-272. 147
16. Salton, G.: *Automatic Text Processing - The Transformation, Analysis and Retrieval of Information by Computer*. Addison-Wesley (1989). 158
17. Sciore, E., Siegel, M., Rosenthal, A.: Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Trans. on Database Systems*, **19**(2), (1994) 254-290. 158

Modeling and Maintaining Multi-View Data Warehouses^{*}

I. Stanoi, D. Agrawal, and A. El Abbadi

Department of Computer Science
University of California, Santa Barbara, CA 93106

Abstract. Data warehouses are designed mostly as centralized systems, and the majority of update maintenance algorithms are tailored for this specific model. Maintenance methods have been proposed either under the assumption of a single view data warehouse, a multi-view centralized model, or a multi-view distributed system with strict synchronization restrictions. We argue that extending this model to a multi-view distributed one, is a practical generalization of the data warehouse system, and the basis for a growing number of applications based on the idea of cooperative views. In this paper we develop a general framework for modeling the maintenance of multi-views in a distributed, decentralized data warehouse, together with an efficient incremental algorithm for view maintenance. To our knowledge, there is no other proposal for a method that incorporates individually and asynchronously updates to different views that are related to each other through derivation dependencies.

1 Introduction

A data warehouse is an example of a storage component that derives, summarizes and integrates information from multiple sources [4], which may be stand-alone databases as well as sites on the Internet. We are interested in analyzing the *distributed* multi-view data warehouse/data marts model, since it can be used to derive information from multiple and diverse data sources that may be geographically dispersed. As the use of views becomes a popular technique for integrating diverse data sources (e.g., data stored in different databases, file-systems, or even Web sites), it can be anticipated that the present centralized model would evolve into a distributed one. We expect that views materialized at different sites may depend not only on base sources, where data is generated, but may also depend on other derived views. This may be due to data security restrictions, requirements on how data flows through an organization, or the sharing of information between companies. The hierarchical arrangement of views also provides the possibility of accessing data at different granularity levels and facilitates operations such as drill-down or roll-up. The distributed multi-view data warehouse is emerging as a design solution for a growing set

^{*} This work was supported in part by NSF under grant numbers CCR-9712108 and EIA-9818320 and IIS98-17432.

of applications, such as storage of messages from newsgroups [5] or cooperative web caches for the Internet. We therefore believe that it is important for the database community to continue to realize the possibilities offered by exploring the distributed multi-view model.

In this paper we develop a general framework for modeling the maintenance of multi-views in a distributed data warehouse together with an efficient incremental algorithm. The key issue is that in a multiple view environment, unlike in the case of the single view (or independent multi-views) model [3,1,12,13,8,6], a view may derive information from the source data via multiple paths through intermediate views. For consistency, the views should reflect the same state of common data sources. Zhuge et al. [11] have proposed using a synchronization processing unit for ensuring consistency, appropriate for a centralized multi-view data warehouses. A decentralized approach has been proposed in [9], based on calculating the cumulative effect of a batch of updates. The consistency criterion there was relaxed, allowing a view to be less restrictive, and derive data from multiple states of a base source. In contrast, our method pertains to decentralized systems, and allows each view to correctly incorporate updates independently. It differs from previous work in its capacity to integrate several advantages in the context of a decentralized, distributed setting: i)an update to a view can be determined solely through communication with the views or data sources from which it directly derives data; ii)no unnecessary delays are imposed due to the grouping of new updates in batches and no centralized coordination is necessary; iii)our method does not require two copies of the data to be stored in the view [8], avoids rematerialization, and queries return the most up to date information available at the view.

In Section 2 we define the multi-view data warehouse system and a corresponding consistency criterion, while Section 3 presents the complexity of incremental view maintenance in such a model. We then develop in Section 4 a decentralized incremental algorithm for updating the information stored by views. This algorithm is further extended to accommodate the grouping of multiple updates generated from the same source. In Section 5 we define a cost formulation for the number of messages exchanged during the updating of a hierarchy of views, and we compare different organizations of dependencies for the same set of views and base sources. A discussion of the proposed algorithms concludes this paper.

2 The Multi-View Data Warehouse Model

2.1 The Hierarchical Model

In the multi-view distributed data warehouse model, data sources and derived views may be stored at different sites. We assume that pairwise communication among sites is both reliable and FIFO. Views are allowed to derive information from base sources as well as from other views.

We first model a distributed multi-view data warehouse/data mart using the notion of a View Dependency Graph [3]. This type of graph has nodes

for sites hosting views and base sources, and the directed edges between nodes indicate the direction of the flow of update messages. An edge directed towards any view S , connects S to one of its ancestors. Figure 1 is an example of a View Dependency Graph where B_1 , B_2 and B_3 are base sources and S_1 , S_2 and S_3 are derived views. S_1 and S_2 are *direct ancestors* of S_3 , and one of their *common ancestor* is B_2 . S_3 is a *direct descendant* of both S_1 and S_2 . The path in the View Dependency Graph determine the update dependencies among different sites. The maintenance of such interdependent views brings out issues that are not present if views are restricted to directly derive data exclusively from base relations. The complexity of a maintenance algorithm arises from the need to correctly propagate updates from base sources to views and *simulate* synchronicity in an asynchronous system.

Colby et al. [3] considered multiple views assumed to be stored at a single site. Hence, their notion of consistency enforces a temporal order on the updates and ensures that views are updated in this order. Since this is too restrictive in a distributed setting, we propose a new consistency criterion, *common base consistency* specifically for distributed multi-view data warehouses. Consider a base source B that executes a sequence of atomic updates (for simplicity, we assume that a single base relation exists per data source). Such an *update* to a base relation, may actually refer to either a single insert or delete operation, or to a set of changes that make up local atomic transactions. These changes result in a sequence of states where we use the notation $\theta_m(B)$ for the m 'th state in this sequence. We now formally define the notion of *Common Base Consistency*.

Definition 1 *Common Base Consistency.* Assume a view V whose state is derived directly from a set of sites $\{S_1, S_2, \dots, S_k\}$ and their states $\{\theta(S_1), \theta(S_2), \dots, \theta(S_k)\}$, where S_i is either a base source or a view. We say that V is common base consistent if for every pair (S_i, S_j) and for every base source B such that $\theta(S_i)$ is derived from state $\theta_m(B)$ and $\theta(S_j)$ is derived from state $\theta_n(B)$, $\theta_m(B)$ must be the same as $\theta_n(B)$.

The above definition simply states that a view V is *common base consistent* if at any time its data reflects at most one state of each base source in the data warehouse. In the example in Figure 1, for common base consistency, any state $\theta(S_3)$ of S_3 can only incorporate states of S_1 and S_2 that reflect the same state of B_2 .

Data at the view is integrated from the sources according to a function, which we assume to be commutative, associative, and distributive. In the case of a data warehouse where base sources are relational databases, the deriving functions generally are operations such as join, select, project, which filter a subset of tuples over which the view may further execute local aggregations. In order to simplify our presentation, we will refer to the deriving function as only a *join* operation. While the commutativity property is necessary for the correctness of our algorithm, the distributivity and associativity play an important role in reducing the communication cost of view maintenance.

Since information about the initial individual tuples is lost through select, project or join operations, multiple entries in the base relation can be mapped

to the same tuple in the resulting view. As a result, the action of a delete operation on a view can lead to an incorrect state. To avoid such inconsistencies, we assume that a count is associated with each tuple that is part of the materialized view [12], and it refers to the number of ways in which a tuple may be derived. The count should be associated with the tuples as soon as they are derived, and included in any updating messages containing the tuples.

2.2 Query Processing during the Update Phase

One of the main challenges in maintaining a data warehouse is to consistently incorporate into views the effects of changes in the source data. Typically, this process is initiated when a view receives information regarding an update to an ancestor that has incorporated new data. Consider a view V that integrates data from sources $S_1, S_2 \dots S_n$, which may be data sources or derived views. Without loss of generality, assume that V is defined as the join operations on $S_1 \bowtie S_2 \bowtie \dots \bowtie S_n$. Each source S_i experiences a possibly empty update due to some change at the base sources (inserts and deletes) ΔS_i . Note that all ΔS_i ($1 \leq i \leq n$) included in the update phase of V must be derived from the same update at a data source. A descendent view V , needs to initiate an *update phase* where it communicates with all its direct ancestors through *update queries* to evaluate, for each ΔS_i , the effects on the remaining ancestors. As a result of V 's update phase, its new state will become $V + \Delta V$, shown below:

$$\Delta V = \sum_n^{i=1} S_1 \dots \bowtie S_{i-1} \bowtie \Delta S_i \bowtie (S_{i+1} + \Delta S_{i+1}) \dots \bowtie (S_n + \Delta S_n)$$

In the above equation $S_1 \bowtie S_2 \dots \bowtie S_i \dots \bowtie S_n$ is the initial state of V . View V needs to formulate update subqueries which are then summed up to calculate the resulting effect on V , ΔV . For common base consistency, each update phase that V undergoes has to reflect the effects of only one update at the base sources.

If all subqueries start with the term of the form ΔS_i , the computation of the different steps in the subquery involves only join operations based on the *change* at one of the views. The messages are therefore small as compared to subqueries that start by forwarding whole relations. Although we assume that V queries its ancestors in the order described by the equation, the only condition we actually impose is that subqueries start with a term of the form ΔS_i .

3 Difficulties in Maintaining a Hierarchy of Views

Most incremental view maintenance algorithms proposed in the literature are tailored for the case of a single view incorporating changes from a single or multiple data sources. In the case when views are allowed to derive updates from other views, there is a need for synchronization of updates between a view and its ancestors. Due to the physical distribution of warehouse views, it is necessary that this synchronization cost be as low as possible. If a view S_3 receives information from views S_1 and S_2 (Figure 1), then the messages received from these ancestors should reflect the same order of updates. Since sites are autonomous,

this assumption is very restrictive. Rather than imposing synchronization on the arrival of messages, our approach allows two views S_1 and S_2 to answer the updating queries of descendent S_3 's update phase as though they did undergo the same order of updates. Unfortunately, in reality the update orders at S_1 and S_2 may be inconsistent with each other. We next illustrate this problem through an example. We use the following notation for updates at a view S_i : $\Delta S_i[\Delta B_j]$

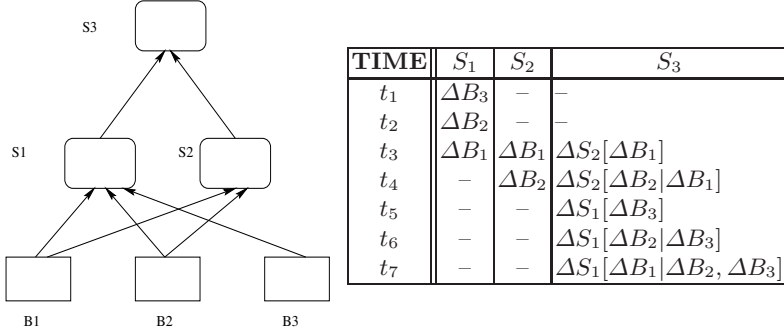


Fig. 1. A Hierarchical Data Warehouse and its View Updates

refers to changes at view S_i as a consequence of the update ΔB_j and $\Delta S_i[\Delta B_j|\Delta B_1, \dots, \Delta B_k]$ refers to an update to a view S_i due to the change ΔB_j with the condition that the local updates resulting from $\Delta B_1, \dots, \Delta B_k$ have already been incorporated. Since the final effect of updates $\Delta B_1, \dots, \Delta B_k$ is cumulative, their ordering in the notation for $\Delta S_i[\Delta B_j|\Delta B_1, \dots, \Delta B_k]$ is not important.

Consider, for illustration purposes, a scenario in which update messages are delivered to the different views in Figure 1 in the order specified. Views S_1, S_2 and S_3 will undergo update phases to calculate one by one the effects of the updates on the local data. View S_1 incorporates ΔB_3 , followed by ΔB_2 and ΔB_1 , and S_2 changes its data to reflect the updates ΔB_1 followed by ΔB_2 . The different update phases that views S_1 and S_2 undergo, involve queries to their direct ancestors, which result in the following changes at the views:

$$\begin{aligned}
 &\text{initial } S_1 = B_1 \bowtie B_2 \bowtie B_3; & \text{initial } S_2 = B_1 \bowtie B_2 \\
 &\Delta S_1[\Delta B_3] = B_1 \bowtie B_2 \bowtie \Delta B_3; & \Delta S_2[\Delta B_1] = \Delta B_1 \bowtie B_2 \\
 &\Delta S_1[\Delta B_2|\Delta B_3] = B_1 \bowtie \Delta B_2 \bowtie (B_3 + \Delta B_3); & \Delta S_2[\Delta B_2|\Delta B_1] = (B_1 + \Delta B_1) \bowtie \Delta B_2 \\
 &\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3] = \Delta B_1 \bowtie (B_2 + \Delta B_2) \bowtie (B_3 + \Delta B_3).
 \end{aligned}$$

Assume that view S_3 decides to calculate first the effect of the change propagated from base source B_1 , then from B_2 followed by B_3 . First note that to ensure common base consistency, S_3 cannot compute the update $\Delta S_3[\Delta B_1]$ until it has received the corresponding changes reflecting the incorporation of ΔB_1 into S_1 and S_2 , i.e., $\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$ and $\Delta S_2[\Delta B_1]$. In order to calculate update $\Delta S_3[\Delta B_1]$, S_3 needs to evaluate:

$$\Delta S_3[\Delta B_1] = \Delta S_1[\Delta B_1] \bowtie (S_2 + \Delta S_2[\Delta B_1]) + S_1 \bowtie \Delta S_2[\Delta B_1]$$

where $\Delta S_1[\Delta B_1]$ reflects a change due to ΔB_1 but not ΔB_2 or ΔB_3 . *This form of $\Delta S_1[\Delta B_1]$ is not the actual update to S_1 , as the update message received from S_1 is $\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$ and hence the update maintenance protocol needs to develop a technique to extract that information with as little overhead as possible.* This example indicates that when an update of the form $\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$ arrives at S_3 , and S_3 is only interested in incorporating the effects of ΔB_1 , enough information must be available at S_3 to extract $\Delta S_1[\Delta B_1]$ from $\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$.

As we will show next, sites also have to be able to compute *states*, not only *changes* to the states. Consider now the update phase that S_3 undergoes in order to integrate the effects of ΔB_2 :

$$\begin{aligned} \Delta S_3[\Delta B_2|\Delta B_1] &= \Delta S_1[\Delta B_2|\Delta B_1] \bowtie (S_2 + \Delta S_2[\Delta B_1] + \Delta S_2[\Delta B_2|\Delta B_1]) \\ &\quad + (S_1 + \Delta S_1[\Delta B_1]) \bowtie \Delta S_2[\Delta B_2|\Delta B_1] \end{aligned}$$

This update phase needs to include an update query to S_1 , based on the state $S_1 + \Delta S_1[\Delta B_1]$, which reflects updates from B_1 but not B_2 and B_3 . In this case the *necessary state is not consistent with the sequence of states that exist at S_1* , and therefore cannot be retrieved from the data stored at S_1 (recall that S_1 incorporated update ΔB_1 after updates ΔB_2 and ΔB_3). Any naive attempt at S_1 to compute this state would involve querying indirect ancestors to retrieve information about earlier states, which in the worst case may propagate to the level of base sources. This example illustrates the need for a site to answer a query based on a state which is not part of its sequence of states. In particular, since S_3 dispatches subquery $\Delta S_2[\Delta B_2|\Delta B_1] \bowtie (S_1 + \Delta S_1[\Delta B_1])$, to S_1 , S_1 should have enough information available either locally or through its ancestors so that it can extract $S_1 + \Delta S_1[\Delta B_1]$ from $S_1 + \Delta S_1[\Delta B_3] + \Delta S_1[\Delta B_2|\Delta B_3] + \Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$.

4 Incremental Maintenance of Multiple Views

In this section we propose an incremental algorithm for maintaining views in a multi-view setting. We first define the storage requirements, then give a detailed explanation of the updating method together with an illustrative example, and finally we discuss some of the properties of the algorithm.

4.1 Storage and Control Structures

Updates are incorporated into a view in an incremental and permanent manner, and two consecutive states differ from one another by exactly one update to a base relation. Each view maintains a materialized state that is a snapshot of a *safe* state as a relational table. To subtract update effects, we require that a view implement a separate storage, the *table of changes*, that includes all the updates following the safe state of the actual materialized view. Although a view is able to calculate older states of its data for user queries, it will also have

access to the most current state based on its actual materialized data as well as the table of changes. The table of changes stores updates in the order they are incorporated at a view, and delete operations are noted as negative inserts. Since this order may not be the same as that requested by descendants trying to update their views, a view answering update queries needs more information to be able to locally exclude the effects of unwanted out-of-order updates. We therefore append a dependency list (referred to as *dep*) to all entries in the table of changes. Each entry is a term returned as answer for an update query, and it reflects one or more changes to different base sources. The dependency list is a set of *identifiers* ΔB_i which indicate that the entry is dependent on changes to the corresponding B_i .

Consider again the example presented in the previous section. The following are the materialized state and table of changes of S_1 and S_2 following their update phases. We also include the dependency lists that correspond to terms in the table of changes.

safe state of S_1 : ($B_1 \bowtie B_2 \bowtie B_3$)

table of changes:

$$\Delta S_1[\Delta B_3] = (\Delta B_3 \bowtie B_2 \bowtie B_1) \text{dep}_{\Delta B_3}$$

$$\Delta S_1[\Delta B_2 | \Delta B_3] = (\Delta B_2 \bowtie B_1 \bowtie B_3) \text{dep}_{\Delta B_2} + (\Delta B_2 \bowtie B_1 \bowtie \Delta B_3) \text{dep}_{\Delta B_2, \Delta B_3}$$

$$\Delta S_1[\Delta B_1 | \Delta B_2, \Delta B_3] = (\Delta B_1 \bowtie B_2 \bowtie B_3) \text{dep}_{\Delta B_1} + (\Delta B_1 \bowtie \Delta B_2 \bowtie B_3) \text{dep}_{\Delta B_1, \Delta B_2} + (\Delta B_1 \bowtie B_2 \bowtie \Delta B_3) \text{dep}_{\Delta B_1, \Delta B_3} + (\Delta B_1 \bowtie \Delta B_2 \bowtie \Delta B_3) \text{dep}_{\Delta B_1, \Delta B_2, \Delta B_3}$$

safe state of S_2 : ($B_1 \bowtie B_2$)

table of changes:

$$\Delta S_2[\Delta B_1] = (\Delta B_1 \bowtie B_2) \text{dep}_{\Delta B_1}$$

$$\Delta S_2[\Delta B_2 | \Delta B_1] = (\Delta B_2 \bowtie B_1) \text{dep}_{\Delta B_2} + (\Delta B_2 \bowtie \Delta B_1) \text{dep}_{\Delta B_1, \Delta B_2}$$

Updates to Base Sources		
B_1 (A,B)	B_2 (B,C)	B_3 (C,D)
(1,2)	(2,4)	(4,1) (3,4)
ΔB_1 (2,2)	ΔB_2 (2,3)	ΔB_3 (3,1)

Updates to Views S_1 and S_2		
S_1 (A,B,C,D)		
safe state (1,2,4,1)		
table of changes	dep	update
–	–	$\Delta S_1[\Delta B_3]$
(1,2,3,4)	$\{\Delta B_2\}$	$\Delta S_1[\Delta B_2 \Delta B_3]$
(1,2,3,1)	$\{\Delta B_2, \Delta B_3\}$	
(2,2,4,1)	$\{\Delta B_1\}$	$\Delta S_1[\Delta B_1 \Delta B_2, \Delta B_3]$
(2,2,3,4)	$\{\Delta B_1, \Delta B_2\}$	
(2,2,3,1)	$\{\Delta B_1, \Delta B_2, \Delta B_3\}$	
S_2 (A,B,C)		
safe state (1,2,4)		
table of changes	dep	update
(2,2,4)	$\{\Delta B_1\}$	$\Delta S_2[\Delta B_1]$
(1,2,3)	$\{\Delta B_2\}$	$\Delta S_2[\Delta B_2 \Delta B_1]$
(2,2,3)	$\{\Delta B_1, \Delta B_2\}$	

Fig. 2. Updates

To show the use of both the storage separation and the dependency lists, we describe below an instance of the illustrative example, according to the above update order at each site. The first update table in Figure 2 includes both the initial states and the insert updates at base relations B_1 with attributes (A, B) , B_2 with attributes (B, C) and B_3 with attributes (C, D) . The second table contains the *safe* state as well as the table of changes of S_1 and S_2 . Assume relation S_1 has attributes (A, B, C, D) , and the attributes of S_2 are (A, B, C) . Dependency lists are associated with the different entries in the table of changes for each view.

Following the update $\Delta S_1[\Delta B_3]$, there are no new tuples to be stored at the view S_1 . The first tuples inserted in the table of changes of S_1 are $(1, 2, 3, 4)$ and $(1, 2, 3, 1)$, due to $\Delta S_1[\Delta B_2|\Delta B_3]$. The remaining tuples in S_1 's table of changes are a result of the last update, $\Delta S_1[\Delta B_1|\Delta B_2, \Delta B_3]$. In the table of changes of S_2 , the tuple $(2, 2, 4)$ is generated by the computation of $\Delta S_2[\Delta B_1]$, while tuples $(1, 2, 3)$ and $(2, 2, 3)$ are due to $\Delta S_2[\Delta B_2|\Delta B_1]$.

The incremental view update model we present is based on the observation that for a view, the effects of incorporating an update ΔB_i over the entries affected by ΔB_j are the same tuples as those reflecting the change ΔB_j over the entries affected by ΔB_i :

Theorem 1. Let a view V undergo updates due to changes at base sources $\Delta B_1, \dots, \Delta B_i, \dots, \Delta B_n$. The exclusion of the updates whose dependency lists include ΔB_i leads to a correct state of V based on updates $\Delta B_1, \dots, \Delta B_{i-1}, \Delta B_{i+1}, \dots, \Delta B_n$.

A proof of Theorem 1 can be found in [10].

All entries that include ΔB_i reflect all and exactly the effects of having incorporated the update ΔB_i , as the latest change. Hence by excluding the entries that include ΔB_i in their attached dependency lists, we **undo** the effects of the update ΔB_i . Referring back to our example, if we want to compute the state of S_1 as if its local data has been affected by only one update ΔB_1 , we have to ignore the effects of ΔB_2 and ΔB_3 from the table of changes. To undo the change due to ΔB_2 , we decide on the set of entries that represent all the effects of ΔB_2 and therefore include ΔB_2 in their dependency lists, i.e., $\{(1, 2, 3, 4), (1, 2, 3, 1), (2, 2, 3, 4), (2, 2, 3, 1)\}$. Since this set already includes all the effects of ΔB_3 , the only entry that is valid for $\Delta S_1[\Delta B_1]$ is $\{(2, 2, 4, 1)\}$.

4.2 Algorithm: Overview

We now present the algorithm for the incremental maintenance of views. In [10] we include in more details the procedure for answering update queries, and outline the steps of a view's update phase.

MultiView Data Warehouse Maintenance:

1) Begin Update Phase. The update phase is triggered by the receipt of a set of changes to the ancestors and their attached dependency lists. Each

view maintains these updates in an *update message queue*. Once a view V has received all messages from its ancestors S_i regarding a particular update ΔB , i.e., $\Delta S_i[\Delta B] \dots$, it initiates the update phase. Since update messages are not synchronized, the update ΔS_i chosen may not be the first in the message queue from S_i . Views that derive data directly from a base source B , can correctly create an identifier for the update ΔB from source B . Since communication is reliable and FIFO between a pair of sites, the views can create this identifier based on the sender and the order in which updates are received from the sender. Views that derive information from other views, receive the identifier in the dependency list attached to the update message.

2) Determine the set of ΔS_i . Once the set of ΔS_i is selected, the updating view must determine the actual form of each ΔS_i , as required by the update phase. A deriving view stores enough information in its message queue to correctly eliminate the effects of unwanted updates ΔB_k from a message $\Delta S_i[\Delta B] \dots, \Delta B_k, \dots$. This is done by eliminating from $\Delta S_i[\Delta B] \dots, \Delta B_k, \dots$ all terms whose dependency lists include the identifier of the updates ΔB_k .

3) Dispatch Queries. As a view V enters the updating phase in order to incorporate the effects of an update ΔB , it dispatches update queries to its direct ancestors. One such query to an ancestor S_j includes the partial result of previously queried ancestors $S_i, \dots, S_{j-1}$, and a list of delayed updates that V received before $\Delta S_j[\Delta B] \dots, \Delta B_k, \dots$. If any of the queried ancestors S_j is a base relation, then the updating view receives an answer based on the current version of S_j , and locally subtracts effects of unnecessary updates. The technique used is similar to the one described in [1]. Otherwise, if the queried ancestor is a view, it is expected to locally calculate the correct answer to the update query (see **QueryServer**), and no additional compensation is needed by the querying view.

4) Refresh the Materialized View. A materialized view is refreshed to incorporate an update from the table of changes only when the new state is considered safe, i.e., when the respective updates are in sequence, and they have been integrated by all the corresponding direct descendents. When updates are included in the actual materialized view, delete operations can be safely incorporated by deleting the appropriate tuples in the relation. As dependency lists are no longer necessary to describe tuples in a *safe* state, they will not be included in the actual materialized views.

End MultiView Data Warehouse Maintenance

Now we consider that a view S_j that is queried for a partial result in the update phase of its descendent V . Assume the current update phase computes the effects due to the changes to a base source B . A view S_j may or may not be asked to include in this set the effects of ΔB , depending on whether S_j is to the right or to the left of S_i (i.e., $j \leq i$ or $j \geq 1$). This information is passed to S_j together with the query request message and the set of updates to be excluded from the computation, *IgnoreSet*.

QueryServer:

1) Compute the Partial Query Answer. View S_j performs the join over its materialized view and the table of changes from which it excludes the terms containing in their dependency lists updates that have not been incorporated yet by the descendent. The entries to be ignored are selected based on the information in the IgnoreSet.

2) Send the Answer to the Updating View. Once a site has computed the correct response to an update query, it appends the appropriate dependency lists and sends the answer to the querying view. Following the termination of the update phase, the updating view V will include the corresponding identifier ΔB in the dependency lists of all new entries calculated during the update phase. After a site incorporates an update, it asynchronously sends this change and the corresponding dependency lists to its descendents.

END QueryServer

In a distributed system, messages from different ancestors can arrive at a view out of order. As the number of messages involved in the updating phase of V does not change, the only difference is in the local computation of the ancestors in answering the update query. In our algorithm, the local elimination of unwanted updates is easily executed during the computation of the deriving function, by ignoring the respective entries in the table of changes. The only additional effort is in scanning the dependency lists associated with the different entries, which is independent of the number of tuples to be ignored. Therefore the local computational effort in answering queries does not depend on the order of updates to be integrated by the querying view. Hence, the reordering of updates is achieved at no additional computational and communication cost and some storage overhead (i.e., storage necessary to maintain dependency information).

4.3 Discussion

The number of messages dispatched during the update phase by a view V with n direct ancestors is $O(n^2)$. That is, in the worst case an update to a base source is propagated to all of V 's direct ancestors, and the effects on each of these ancestors must be calculated with respect to all other direct ancestors. The number of messages can be limited to $O(n)$, with the tradeoff of increasing their size. The update subqueries can be processed concurrently, through the dispatch of only two queries to each of the ancestors. Depending on the definition of the view, the reduction in the number of messages might be advantageous in some systems. Since the updating algorithm is not dependent on the order of query, it can be easily adapted to the two-phase querying method. A recent paper ([7]) proposed an ordering of the subqueries, that would optimize the cost of their computation. However, the results described do not hold in the case of a view deriving data from two or more sites that have a common ancestor. Since the purpose of this paper is to introduce a maintenance algorithm for a general hierarchy of views, we do not enforce this reordering of the subqueries.

Further optimizations can improve the update queries, by choosing an optimal order over the different terms of a subquery [2]. Since we are analyzing a general hierarchical model, we cannot make further assumptions on the query optimizations. The challenge we are addressing in this paper is to perform these join operations with the appropriate state, i.e., one that reflects the correct set of updates according to the definition of common base consistency. Note that, although the correct updating of distributed views in a decentralized setting is an important and challenging problem, this issue has not been addressed until now.

The number of messages exchanged depends on the structure of the updating query, as well as on the clustering of updates. If updates to base sources are frequent, then the single-update incremental maintenance algorithm may be inefficient as compared to an algorithm that groups updates in batches. The maintenance algorithm we proposed in Section 4.2 is flexible enough to easily integrate the grouping of updates so that, for an updating view V , each ΔV computes the effects of a batch of updates to a base source B . Previous to dispatching the update query, V groups the incoming update messages from its ancestors S_i in separate lists of $\sum \Delta S_i(\Delta B)$, for changes propagated from base source B . Then for any S_i and S_j ancestors of V , if both S_i, S_j derive from a source B , then a change ΔB must be either in both $(\sum \Delta S_i(\Delta B) \text{ and } \sum \Delta S_j(\Delta B))$ or in neither of them. The general order of query is the same as discussed in the single-update case, the only difference being in the replacement of ΔS_i with $\sum \Delta S_i$.

5 Message Cost Analysis

In this section we present a message cost formulation specific to the updating of a hierarchy of distributed views. Based on this cost formulation, we compare different hierarchical structures of the same set of views and base sources.

5.1 Cost Formulation

In general, we can express all dependencies within a hierarchy of views and base sources with the help of a simple binary matrix. There are two types of dependencies that play an important role in the number of messages exchanged during a view's update phase: dependencies on direct ancestors, and dependencies (direct or indirect) on base sources. Although the second type of information can be inferred from the first, we explicitly separate the two and represent them in matrices A and B . A row i in a dependency matrix summarizes the dependencies of a view indexed S_i . The columns in matrix A , describing dependencies on base sources, correspond to each of the sources. The second matrix, B , has $[\text{number of base sources} + \text{number of views}] \times \text{columns}$, for all the possible direct dependencies. For our example, (Figure 1), the matrices are the following:

matrix A				matrix B						
	B_1	B_2	B_3	B_1	B_2	B_3	S_1	S_2	S_3	
S_1	1	1	1	S_1	1	1	1	0	0	0
S_2	1	1	0	S_2	1	1	0	0	0	0
S_3	1	1	1	S_3	0	0	0	1	1	0

Let $NrUpdates_{B_i}$ be the number of batches or single updates at B_i , $NrViews$ and $NrSources$ be the number of views in the hierarchy and, respectively, the number of base sources.

If $U(i)$ = number of subqueries dispatched by view S_i during its update phase, then $U(i)$ is the sum of all possible dependencies on a base source, times the number of updates to the corresponding base source.

$$\bullet U(i) = \sum_{j=1}^{NrSources} (NrUpdates_{B_j} * (\sum_{k=1}^{NrViews} (B[i, (k + NrSources)] \cap A[k, j]) + B[i, j]))$$

For a view S_k , $(B[i, (k + NrSources)] \cap A[k, j]) = 1$ if view S_i derives data directly from S_k and the update B_j is one of S_k 's ancestors. Considering again our example, the number of queries computed during the update phases of views S_1 , S_2 and S_3 are:

$$U(1) = NrUpdates_{B_1} + NrUpdates_{B_2} + NrUpdates_{B_3}$$

$$U(2) = NrUpdates_{B_1} + NrUpdates_{B_2}$$

$$U(3) = NrUpdates_{B_1} * 2 + NrUpdates_{B_2} * 2 + NrUpdates_{B_3} * 1$$

The number of queries differs from the number of messages exchanged, as a query that calculates the effects of a change at one of the ancestors involving messages to all the other direct ancestors. That is, a query involves $(n-1)$ messages, where n is the number of direct ancestors. Let $M(i)$ be the number of queries dispatched to compute all the update queries in a view S_i 's update phase. Then $M(i)$ is tightly related to $U(i)$ and the number of direct ancestors of S_i , $NrAn_{S_i}$. The value of $NrAn_{S_i}$ is easily calculated as the sum of the entries (which store either a "1" or a "0") in matrix B , row i .

$$\bullet M(i) = (NrAn_{S_i} - 1) * U(i)$$

$$\bullet M(i) = (NrAn_{S_i} - 1) * \sum_{j=1}^{NrSources} (NrUpdates_{B_j} * (\sum_{k=1}^{NrViews} (B[i, (k + NrSources)] \cap A[k, j]) + B[i, j]))$$

$$M(1) = NrUpdates_{B_1} * 2 + NrUpdates_{B_2} * 2 + NrUpdates_{B_3} * 2$$

$$M(2) = NrUpdates_{B_1} * 1 + NrUpdates_{B_2} * 1$$

$$M(3) = NrUpdates_{B_1} * 2 * 1 + NrUpdates_{B_2} * 2 * 1 + NrUpdates_{B_3} * 1 * 1$$

Another measure for the number of messages exchanged during the updating of the views in the hierarchy, is the *responsibility*, R , of each view or base source. For a site S_i , $R(i)$ is the number of subqueries dispatched by a descendent of S_i , in which S_i is asked to perform some computation.

$$\bullet R(i) = \begin{cases} \sum_{j=1}^{NrViews} (B[j, i + NrSources] * U(j) - \sum_{k=1}^{NrSources} (A[i, k] * NrUpdates_{B_k})) & \text{if } S_i \text{ is view} \\ \sum_{j=1}^{NrViews} (B[j, i] * U(j) - NrUpdates_{B_i}) & \text{if } S_i \text{ is a base source} \end{cases}$$

$$R(B1) = U(1) - NrUpdates_{B_1} + U(2) - NrUpdates_{B_1}$$

$$R(B2) = U(1) - NrUpdates_{B_1} + U(2) - NrUpdates_{B_1}$$

$$R(B3) = U(1) - NrUpdates_{B_1}$$

$$R(1) = U(3) - (NrUpdates_{B_1} + NrUpdates_{B_2} + NrUpdates_{B_3})$$

$$R(2) = U(3) - (NrUpdates_{B_1} + NrUpdates_{B_2})$$

$$R(3) = 0$$

Note that

$\sum_{i=1}^{NrViews+NrSources} U(i)$, $\sum_{i=1}^{NrViews+NrSources} M(i)$ and $\sum_{i=1}^{NrViews+NrSources} R(i)$ represent the different cost evaluations for the whole hierarchy of views and base sources, and $\sum_{i=1}^{NrViews+NrSources} M(i) = \sum_{i=1}^{NrViews+NrSources} R(i)$.

5.2 Tall or Flattened Hierarchy?

Based on the cost model proposed, we analyze the behavior of a given set of views and base sources in a spectrum of hierarchical structures. The hierarchy of views $S_1, S_2, S_3, S_4, S_5, S_6$ and S_7 in Figure 3 illustrates one end of the spectrum, in which a view is placed in the highest level of the hierarchy while is still able to derive the necessary information. At the other end of the spectrum is the flat arrangement of the views, where all views derive their data directly from base sources. An intermediary structure is also shown in Figure 3, where view S_7 is replaced by S_7' , now deriving data from views S_1, S_2, S_3 and S_4 . The

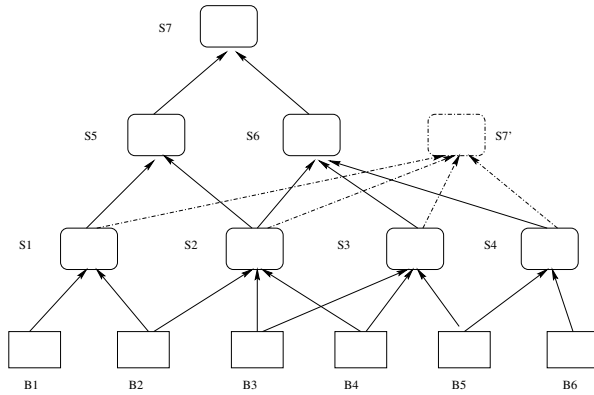


Fig. 3. Three level hierarchy (tall) and two level hierarchy (middle)

views in the three different hierarchies are similarly defined over the base sources. However, the total responsibility of the system, $\sum_{i=1}^{NrViews+NrSources} R(i)$, and therefore the total number of messages exchanged, $\sum_{i=1}^{NrViews+NrSources} M(i)$, differs from one case to another. Figure 4(a) shows that, for equal frequencies of updates at the sources, the total number of messages exchanged during the updating of the views increases when the height of the hierarchy is reduced. If only one update is performed at each base source, then the total responsibility on the system for the three cases is: 46, 67 and 78 messages for the tall, middle and flat hierarchy. These numbers increase linearly with the number of individual updates (or batches) to the sources if the distribution of updates over the base sources remains the same.

The base sources in the cases shown in Figure 3 have the same set of descendants. Therefore, the responsibility load on the sources remains the same. However, as seen in Figure 4(b) by comparing the flat and the tall hierarchies of the same structure, if the number of views directly dependent on base sources increases, then the total responsibility $\sum_{i=1}^{NrSources} R(i)$ of the base sources increases significantly. The fact that tall hierarchies impose a reduced communication and execution load on the sources and in general on the system, supports the idea that, when possible, it is advantageous to allow derivation dependencies between different views.

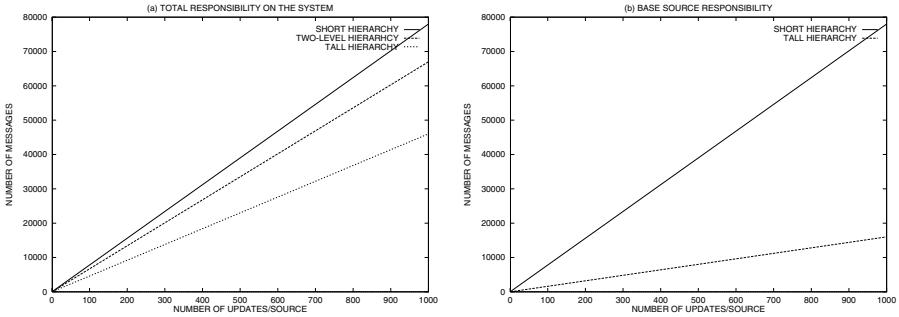


Fig. 4. Comparison of message cost between the tall, two-level and short hierarchy

6 Conclusion

Several maintenance algorithms have been proposed for the data warehouse model. They are specific to one view, or a set of views that have no interdependencies. For multi-view warehouses, previously proposed methods either include a synchronization module, or required a notion of synchronized global time. Both of these requirements are too restrictive in a distributed setting, which is the primary application area of multi-view warehouses, i.e., data marts. In this paper, we first developed a notion of consistency for distributed multi-view data warehouses. We then developed an incremental algorithm for asynchronously incorporating updates into views. Our approach requires the maintenance of a stable state separate from a table of changes. This table of changes gives sites the flexibility of incorporating updates in different orders, and at the same time allow them to respond to queries requiring alternate orders of execution, without significant computational and storage overhead (simply checking the dependency lists). Moreover, local computation of ancestors in answering a view's updating query does not increase if reordering of messages at the view becomes necessary. This decentralized approach then allows sites to make independent decisions on when to incorporate an update, unlike a synchronization module

based approach [11] where the order of updates to all views must be the same, thus imposing unnecessary delays when incorporating new updates. We further extend the single-update maintenance algorithm to an algorithm based on computing changes in batches, which is more appropriate for systems that support a high frequency of updates to the base sources. After developing a cost model for the update messages through the hierarchy, we argue that, if possible, views should be defined to derive data from other views rather than base sources, thus reducing the communication and execution load on the sources.

References

1. D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance in Data Warehouses. In *Proceedings of the 1997 ACM International Conference on Management of Data*, pages 417–427, May 1997. 162, 169
2. Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. In *Proceedings of the ACM PODS*, pages 34–43, June 1998. 171
3. Latha S. Colby, Akira Kawaguchi, Daniel F. Lieuwen, and Inderpal Singh Mumick. Supporting Multiple View Maintenance Policies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 405–416, May 1997. 162, 162, 163
4. A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Bulletin of the Technical Committee on Data Engineering*, 18(2):3–18, June 1995. 161
5. H. Gupta and D. Srivastava. The Data Warehouse of Newsgroups. In *Proceedings of the Seventh International Conference on Database Theory (ICDT)*, pages 471–488, January 1999. 162
6. N. Huyn. Multiple-View Self-Maintenance in Data Warehousing Environments. In *Proceedings of the International Conference on Very Large Data Bases*, pages 26–35, August 1997. 162
7. W. Labio, R. Yerneni, and H. Garcia-Molina. Shrinking the Warehouse Update Window. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 383–394, May 1999. 170
8. D. Quass and J. Widom. On-line Warehouse View Maintenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–404, May 1997. 162, 162
9. I. Stanoi, D. Agrawal, and A. El Abbadi. Weak Consistency in Distributed Data Warehouses. In *Proceedings of the International Conference of Foundations of Data Organization*, pages 107–116, November 1998. 162
10. I. Stanoi, D. Agrawal, and A. El Abbadi. Modeling and Maintaining Multi-View Data Warehouses. Technical report, Department of Computer Science, University of California at Santa Barbara, <http://www.cs.ucsb.edu/ioana/>, 1999. 168, 168
11. Y. Zhuge, J. L. Wiener, and H. Garcia-Molina. Multiple View Consistency for Data Warehousing. In *Proceedings of the 1997 IEEE International Conference on Data Engineering*, pages 289–300, April 1997. 162, 175
12. Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View Maintenance in a Warehousing Environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 316–327, May 1995. 162, 164

13. Yue Zhuge, Hector Garcia-Molina, and Janet L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, pages 146–157, December 1996. 162

Object Views through Search Views of Web Datasources

Zoé Lacroix

Data Logic division, Gene Logic Inc.
2001 Center St, Suite 600, Berkeley CA 94704, USA
zoe@genelogic.com

Abstract. Web datasources usually allow a restricted access (through CGI calls) and their output consists of generated HTML documents. Unfortunately, in many cases the data they provide happen to be available only on the Web. In this paper, we describe a system based on a Web wrapper combined with an object multidatabase system that enables the user to query Web datasources as well as other datasources with an OQL-like query language. The approach is based on two successive views of the Web datasource: a *search view* characterizing its access and an *object view* extending the restricted search view with respect the extraction capabilities of the wrapper. Both views are non-materialized to provide the user with up-to-date data. Our approach has been developed and demonstrated as part of the multidatabase system supporting queries via uniform Object Protocol Model (OPM) interfaces.

1 Introduction

In each domain, accessing the relevant data, combining datasources and coping with their distribution and heterogeneity is a tremendously difficult task. Datasources are heterogeneous and often do not provide the user with a real query language. Web biomedical datasources such as LENS (Linking Expressed Sequence Tags and their associated Name Space) datasource [LEN] or Saccharomyces Genome Database [SGD] provide the biologists with a necessary complement of information to other (local or remote) sources: flat files such as GenBank or SwissProt or relational databases. Web datasources are mostly textual and provide restricted search facilities. Their structure varies from ASN.1 data exchange format to poorly structured HTML format.

The success of database systems invites users to expect the best: the ability to query any source with a real rich query language such as SQL or OQL. To satisfy these users, mediation or multidatabase systems have been proposed by the database community. A multidatabase system requires the user to explicitly mention in the query the sources that have to be queried, whereas a mediator selects the source(s) for the user. Both approaches require integration of sources generally based on the notion of a *view*. Systems such as TSIMMIS [GMPQ⁺97], HERMES [SAB⁺99], OPM multidatabase system [CKMS98] or BIOKLEISLI [BCD⁺98] define the integrated model as a view of the sources

(global as view approach). Others such as Information Manifold [LRO96] consider the sources as views on the global model (local as view approach). The global approach usually makes the process of evaluating queries (as a translation from queries onto the integrated schema into queries on the sources) easier. The local approach models each datasource independently of the other and changes can be handled more easily, on the other hand, query evaluation is more complex.

Both local and global approaches are restricted by the available access to Web sources. Indeed, the wrapping component of the system has the task to send queries to the Web sources (usually CGI calls) and process the output documents to extract the result of the query. Each possible CGI call to access the Web datasource is another (dramatically restricted) view of the source. The provided accesses constitute a *search view* of a Web datasource. Any database view of a Web datasource must be designed with respect to its search view and the extraction capabilities of the system. The process of evaluation of a query has to go through two steps, rewrite the query into a *search query* (on the search view) and into an *extraction query* (on the retrieved documents).

In this paper, we describe the Web wrapper implemented under the OPM multidatabase system [CKMS98], based on the Object Protocol Model (OPM) [CM95] to design object views [CKMS97] of the sources and its query language OPM*QL, an OQL-like query language. The OPM Web wrapper is composed of a retrieval component and a XML engine. The former generates the search query, while the latter evaluates the extraction query upon retrieved documents. The next section defines the successive views and describes the architecture of the system with examples. In Section 3, we focus on query evaluation and, in particular, we address the issue of generating search queries for conjunctive OPM queries and general OPM queries. We conclude with Section 4.

2 Successive Views of Web Datasources

We first define the two-step process to design an object view of a Web datasource. We describe the architecture of our implementation in Section 2.2.

2.1 Search and Object Views

In general, the task of wrapping data from a database system to another is a difficult task when the data model and query language of both sides are known. Web sources are semi-structured [Abi97, Bun97] and their partially unknown structure enhances the difficulty of wrapping. Moreover, search facilities available for Web sources are rather poor. Each Web datasource has entry points (keys in a CGI form, for example) that allow to retrieve documents. A view of a Web datasource is restricted by both the search facilities provided by the source and the extraction capabilities of the system (themselves restricted by the format of the output returned by the source).

Search View of a Web datasource Consider the LENS (Linking Expressed Sequence Tags and their associated Name Space) datasource [LEN], created and maintained at the Computational Biology and Informatics Laboratory at the University of Pennsylvania. It links and resolves the names and identifiers (IDs) of clones and expressed sequence tags (ESTs) generated in the I.M.A.G.E. Consortium/WashU/Merck EST project [EST]. The name space includes library and clone IDs and names from I.M.A.G.E. Consortium, EST sequence IDs from Washington University, sequence entry accession numbers from dbEST/NCBI, and library and clone IDs from GDB.



Fig. 1. Successive views of LENS datasource.

The access to LENS is done through a form (see center part of Figure 1) corresponding to a single CGI call `cgi-bin/lens/viewCloneId.perl?` and several keys such as `I.M.A.G.E. ID`, `I.M.A.G.E. Name` or `GDB ID`, etc. Any call to the source with a valid parameter returns a document such as described in the right part of Figure 1.

We first define the notion of *search attribute*. Intuitively, search attributes correspond to the search facilities provided by the Web datasource. For example, a search attribute for the LENS Web datasource is `image_id`, since it is possible to retrieve a document (see right of Figure 1) with the CGI call `cgi-bin/lens/viewCloneId.perl?all.types.type=I.M.A.G.E.+ID&all.types=` and a valid parameter. The document is the complete description of a clone. Therefore, `image_id` is a search attribute for class `Clone` of datasource LENS.

A search attribute is a key in a CGI form to access a datasource such that the key returns a full description of a single instance with a valid parameter. When the parameter is not valid, no description of an instance is returned. Note that a search attribute is currently a single-valued key attribute since to a value (a valid parameter in the call) must correspond a single instance. An extension of this approach to allow the use of search facilities that return several instances is under implementation.

A *search view* of a Web datasource is the interface between the Web datasource and the system designed with respect to both the search facilities provided by the datasource and the extraction capabilities of the system. A search view is an object view with the following properties.

- (1) each object class corresponds to an output template of a CGI program.
- (2) each object class has at least one *search attribute*.

We represent a search view for a Web datasource in a *Web MAP file* (see Figure 2). The Web MAP file represents the search facilities listing search attributes and their corresponding CGI call. It also lists for each CGI call the parser for its output.

```
DATASOURCE LENS
CLASS Clone
WRAPPER perl wrapper.perl
SEARCH ATTRIBUTE image_id
    CGI http://www.cbil.upenn.edu/cgi-bin/lens/viewCloneId.perl?all
        _types.type=I.M.A.G.E.+ID&all_types=
    PARSER LensClone
SEARCH ATTRIBUTE gdb_id
    CGI http://www.cbil.upenn.edu/cgi-bin/lens/viewCloneId.perl?all
        _types.type=GDB+ID&all_types=
    PARSER LensClone
```

Fig. 2. Web MAP File

In the search view of LENS given in Figure 2, LENS has a single object class `Clone` which has two search attributes `image_id` and `gdb_id`¹. The search attributes respectively correspond to the CGI calls `viewCloneId.perl?all_types.type=I.M.A.G.E.+ID&all_types=` and `viewCloneId.perl?all_types.type=GDB+ID&all_types=`. `LensClone` is the parser for the returned HTML documents. Both search attributes are keys of the same CGI program, therefore only one parser is needed.

Object View of a Web datasource The design of the object schema of the Web datasource follows the search view and is straightforward. Each class of the

¹ We restrict the search view to two search attributes to simplify this presentation.

search view is a class of the object view. The attributes defined at a class in the object view are search attributes and extracted attributes. An object view of a Web datasource respects the following properties.

- (1) each object class is a class in the search view.
- (2) each search attribute of the search view is an attribute of the corresponding class of the object view.
- (3) the value of all attributes of a class of the object view can be extracted from the returned document.

Our approach is page-oriented: each CGI call associated with a search attribute musts return the full description of an object of the object class. The retrieved documents are more or less formatted in HTML syntax. In many cases, the returned document is a textual document in a simple HTML template for display. Our approach currently handles the semi-structure of returned documents by ignoring the unexpected attributes and corresponding data. The object identifier is created with the value of one of the search attributes of the search view of the datasource.

The object view of the LENS datasource consists of a class `Clone` and attributes `image_id`, `gdb_id`, `clone_name`, `date_of_entry`, etc. (see left side of Figure 1). The identity of objects of class `Clone` is built with the value of attribute `image_id`.

2.2 Architecture

We briefly present some features of the OPM Multidatabase Query System (MQS) based on the OPM model [CM95]. It follows the ODMG standards [Ba97] with objects identified by a unique object identifier, qualified by attributes and classified into classes. Attributes are either simple or tuple of simple attributes and classes are organized within a sub-classing relationship. OPM*QL is an object-oriented query language similar to OQL [Ba97]. We do not discuss any of the multidatabase aspects of the system in the paper and invite the reader to read [CKMS98] for a complete description.

Suppose that a user wants to know the name of a clone identified by “25002” in I.M.A.G.E. datasource, the date of its entry in the datasource and its corresponding identifier in GDB. The user asks the corresponding OPM query (see Figure 4) on the OPM schema given in Figure 1. As illustrated in Figure 3, the OPM query is processed by MQS and sent to the Web wrapper. The retrieval component of the Web wrapper generates a search query corresponding to the CGI call `viewCloneId.perl?all_types.type=I.M.A.G.E.+ID&all_types=25002`, to retrieve a document where the value of attributes `clone_name`, `gdb_id` and `date_of_entry` are extracted by the XML engine of the Web wrapper and returned to MQS as a record of values (see Figure 5).

The latter query is equivalent to a single manual browsing query onto the Web site of the datasource consisting in filling out the form and reading the resulting document. However, wrapping a Web datasource into an object database view enables the user to ask more complex queries such as the one given in Figure 6.

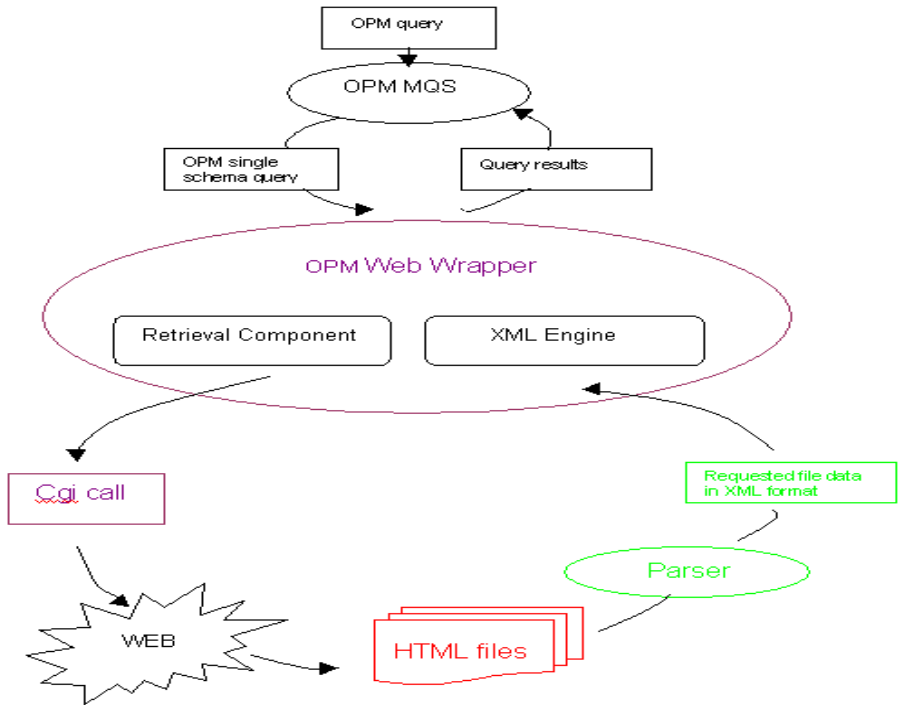


Fig. 3. Architecture of the OPM Web wrapper.

This time, the retrieval component of the Web wrapper generates two search queries to retrieve the description of the clones respectively identified by "25001" and "25002" and the XML engine generates an extraction query to extract the values of `clone_name`, `image_id` and `sex`.

The retrieval component processes the query sent by MQS and generates the CGI calls reading the Web MAP file and file names to cache the returned documents. It returns the file names to be processed to the XML engine. The set of calls sent to the Web datasource to answer a query Q is denoted by $\text{Call}(Q)$. The set of all retrieved objects to answer a query Q is $\text{Retrieve}(Q)$. Intuitively, it corresponds to all documents that have to be retrieved from the datasource in order to answer the query. The cardinals of the two sets are respectively denoted by $\#\text{Call}(Q)$ and $\#\text{Retrieve}(Q)$. Two distinct calls may retrieve the same HTML document (and thus the description of the same object). For instance, the two following CGI calls `viewCloneId.perl?all.types.type=I.M.A.G.E.+ID&all.types=25002` and `viewCloneId.perl?all.types.type=GDB+ID&all.types=397349` retrieve the same HTML document.

```

SELECT  clone_name = @c.clone_name,
        gdb_id = @c.gdb_id,
        date_of_entry = @c.date_of_entry
FROM    @c in LENS:Clone
WHERE   @c.image_id = "25002"

```

Fig. 4. An OPM*QL query

```

clone_name = "am483d07"
gdb_id = "397349"
date_of_entry = "May 19 1995"

```

Fig. 5. Output of the OPM*QL query

As soon as it is retrieved from a Web source, a document is parsed and all the data it contains (the value of all attributes of the OPM class) is extracted and cached in XML format. The XML engine evaluates the extraction query against the cached documents to build the output and to return it to MQS. The current assumption for Web datasources is that the whole description of each OPM object should be available in a cached XML document. Therefore, the output should be complete after processing all cached XML documents. In [Lac99], we describe the process and explain how it can be extended to send other calls to complete the output.

In the following section we detail how the query processing is actually performed to retrieve the set $\text{Retrieve}(Q)$ for each query Q .

3 Query Evaluation

Query evaluation is performed in three successive steps. First MQS simplifies and splits the input query. Secondly, each sub-query is sent to the wrapper to retrieve documents and extract the data. Finally, MQS evaluates the query over the data sent back by the wrapper. We first describe the query splitting performed by the MQS query processor. In a second section, we address the issue of retrieving documents for conjunctive queries. In a last section, we describe the process for general OPM queries allowing conjunction as well as disjunction and negation in the **WHERE** clause.

3.1 Query Splitting

The OPM multidatabase system processes the query, simplifies it (removing path expressions) and splits it in such a way that each sub-query consists in retrieving only one set of objects of an OPM class. After splitting, it successively sends each sub-query to the wrapper.

```

SELECT  clone_name = @c.clone_name
FROM    @c in LENS:Clone
WHERE   @c.image_id = "25001"
        OR  (@c.image_id = "25002"
        AND  @c.sex = "Female")

```

Fig. 6. A complex OPM*QL query

```

Q0: SELECT  description = @l.description
FROM      @l in SACCH:Locus,
          @s in SACCH:Sequence
WHERE     @s.sequence_name = "YCR097W"
AND       @s.locus = @l.locus_name

```

Fig. 7. A query to SACCH

The order of submission of the sub-queries resulting from the processing is fundamental. Consider the *Saccharomyces Genome Database* [SGD], at the Department of Genetics at Stanford University and its search view consisting of two classes *Locus* and *Sequence* respectively associated with the search attributes *locus_name* and *sequence_name*. If a users asks the query Q0 given in Figure 7 against the object view of SGD, Q0 is split into two sub-queries Q1 and Q2 as illustrated in Figure 8. Query Q1 clearly has to be asked first in order to ask query Q2 with the extracted value “a1” corresponding to the search attribute *locus_name* of class *Locus*.

<pre> Q1: SELECT @s.locus = _1 FROM @s in SACCH:Sequence WHERE @s.sequence_name = "YCR097W" </pre>	<pre> Q2: SELECT description=@l.description FROM @l in SACCH:Locus WHERE @l.locus_name = _1 </pre>

Fig. 8. Sub-queries generated from Q0

The usual evaluation of a query such as Q1 by MQS consists in (1) returning all objects of class *Sequence* in datasource SACCH, (2) filtering them to only retain the ones with *sequence_name* of value "YCR097W", and (3) selecting from the remaining the value of attribute *locus*. Clauses FROM, WHERE and SELECT are successively processed. We will see that accessing Web sources requires a different approach. Indeed, partial evaluation is performed by the wrapper. The output returned by the wrapper (precisely its XML engine) is a flat record or a set of flat records. To answer query Q1, each record is a pair [*locus*, *sequence_name*] for an object of class *Sequence* in datasource SACCH.

The multidatabase system creates the object identity with the value of a search attribute assigned as a key for the OPM class, evaluates the query and submits the next sub-query to the wrapper, if any. For each record returned by the XML engine, it extracts the value corresponding to the template to submit the next query. In our example, the returned value for the template `@s.locus = _1` is “a1”. If there several possible substitutions, the queries are successively submitted with an incremented name such as `Q2#1`, `Q2#2`, etc. When all sub-queries are answered, it returns the output to the user.

3.2 Evaluation of Conjunctive Queries

A conjunctive query `Q` submitted to the wrapper requires to retrieve a unique object (`Retrieve(Q)` is a singleton). Indeed, we assume that search attributes are single-valued and keys. Therefore, only one CGI call is necessary to retrieve the document describing the expected object.

To each query `Q` we associate a *search value* that expresses the ability of the retrieval component to build a CGI call for this query. In our approach, a CGI call corresponds to a *search condition*. A conjunctive query has a search value `true` iff it contains at least a search condition.

For a query against a class `Class` of a data source `DS`, a search condition is an atomic expression of the form `@o.attribute = string(v)` or `string(v) = @o.attribute` where `attribute` is a search attribute of class `Class` in a data-source `DS` and `v` is a string non null. An extension of the component to consider other types as well as the matching operator for strings is under implementation.

The search value of a query is assigned as follows. A search condition has a search value `true` when any other atomic expression has a search value `false`. The search value of a conjunction of expressions is `true` if at least one of them is of true search value.

An expression `exp1 AND exp2` such that `exp1` and `exp2` have respectively true and false search values is interpreted the following way. The document validating `exp1` is retrieved when no document for `exp2` is retrieved. It does not mean that no corresponding objects exists in the Web datasource, it only means that no such document is accessible with this condition. The retrieval set is a singleton, therefore only one object must be retrieved. Expression `exp1` is used to retrieve the object, when expression `exp2` is later evaluated by the database system as a usual condition over the retrieved object. Since the expression `exp1` is already validated, partial evaluation has been performed at the level of the wrapper.

A query `Q` of the form `SELECT ... FROM @o in DS:Class WHERE exp` has a search value `true` iff `exp` is of true search value. Assuming the metadata concerning class `Class` of datasource `DS` is available in the Web MAP file, the call corresponding to query `Q` of search value `true` is built with `ConjCall(exp)` as described in Figure 9.

This function is optimized with accessing the cache and checking whether or not the description of the expected object is already cached. The latter reduces the number of calls actually sent to the Web datasource. In order to always

1. if `exp = @o.attribute = string(v)` or `exp = string(v) = @o.attribute` where `attribute` is a search attribute of class `Class` in a datasource `DS` and `v` is a string non null, then send the corresponding CGI call with parameter `v`.
2. if `exp = exp1 AND exp2`, if `exp1` is of true search value then `ConjCall(exp)=ConjCall(exp1)`, otherwise `ConjCall(exp)=ConjCall(exp2)`.

Fig. 9. Algorithm to build CGI call for conjunctive queries

retrieve up-to-date data, the cache is emptied at the end of each querying session (when the OPM*QL query is answered to the user). It is important that data is not cached for a long period of time, because these datasources are often updated and our users want to access the most recent data.

3.3 Dealing with Disjunction and Negation

In the general case, several calls have to be built and sent to retrieve the set `Retrieve(Q)`. Indeed, to answer a query such as given in Figure 10, the wrapper must retrieve two objects: the one identified in I.M.A.G.E. datasource by “25002”, and the one identified by “1303251” in GDB.

```
SELECT  clone_name = @c.clone_name,
        date_of_entry = @c.date_of_entry
FROM    @c in LENS:Clone
WHERE   @c.image_id = "25002"
        OR   @c.gdb_id = "1303251"
```

Fig. 10. A disjunctive query

An expression `exp1 OR exp2` such that `exp1` and `exp2` have respectively true and false search values fails. Indeed, the condition expressed by `exp2` could not be evaluated on the Web source. Generally, the search value of a conjunction of expressions is **true** if one of the expressions is of true search value. The search value of a disjunction of expressions is **true** if both of them are of true search value. The negation of an expression is of search value **false**. A query `Q` is of search value true, if its `WHERE` clause is of true search value.

The retrieval component generates one or more retrieval call(s) from query of search value **true** (an easy induction proves that a query `Q` is of search value true iff `Call(Q) ≠ ∅`). The retrieval component of the OPM Web wrapper processes a query of search value **true**, extracts the metadata from the Web MAP file and builds the call(s) and the output name(s) of the retrieved file(s). Before sending the call to the Web source, it checks whether or not the document(s) are already cached. It returns the output file name(s) to the XML engine.

1. if $\text{exp} = @o.\text{attribute} = \text{string}(v)$ or $\text{exp} = \text{string}(v) = @o.\text{attribute}$ where **attribute** is a search attribute of class **Class** in a datasource **DS** and **v** is a string non null, then $\text{Call}(\text{exp}) = \{ \text{the corresponding CGI call with parameter } v \}$.
2. if $\text{exp} = \text{exp1 AND exp2}$, if exp1 is of true search value then $\text{Call}(\text{exp}) = \text{Call}(\text{exp1})$, otherwise $\text{Call}(\text{exp}) = \text{Call}(\text{exp2})$.
3. if $\text{exp} = \text{exp1 OR exp2}$, then $\text{Call}(\text{exp}) = \text{Call}(\text{exp1}) \cup \text{Call}(\text{exp2})$.
4. if $\text{exp} = \text{NOT exp1}$, then $\text{Call}(\text{exp}) = \emptyset$.

Fig. 11. Algorithm to build CGI calls for OPM queries

4 Conclusion

The OPM multidatabase system [CKMS98] is now extended with wrappers for Web datasources, implemented by the Data Logic group at Gene Logic. Each Web datasource is described in a Web MAP file that allows to build a search view. All wrappers use a common XML engine which parses XML documents generated by parsers.

Our actual implementation can be extended to make a better use of the resources of Web datasources.

- Some datasources are available in different sites (mirror sites) and the retrieval server could take advantage of them.
- An extension of the definition of search conditions to handle other types or the use of the match operator is under implementation.
- The query splitting step can be optimized and several solutions are currently being studied.

The use of a multidatabase system considerably extends the approach with the ability to combine data retrieved from the Web with data extracted from other sources. The use of a multidatabase system instead of a standard one obviously adds more advantage to the approach. Not only these datasources can be queried with a real query language but the user is able to ask queries over several sources explicitly linked in a global OPM schema.

Acknowledgments

The authors wish to thank the Data Logic group at Gene Logic for their useful comments when designing and implementing the wrapper for Web sources. In particular Anthony Kosky is thanked for answering all questions regarding the OPM multidatabase system and its implementation.

References

- Abi97. S. Abiteboul. Querying semi-structured data. In *Proc. of Intl. Conf. on Database Theory*, pages 1–18, Delphi, Greece, January 1997. LNCS 1186, Springer. 177
- Ba97. D. Bartels and al. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, San Francisco, 1997. 180, 180
- BCD⁺98. P. Buneman, J. Crabtree, S. Davidson, V. Tannen, and L. Wong. BioK-leisli. *BioInformatics*, 1998. <http://www.cbil.upenn.edu/K2/>. 176
- Bun97. P. Buneman. Semistructured data. In *Proc. ACM Symp. on Principles of Database Systems*, Tucson, 1997. Invited tutorial. 177
- CKMS97. I.A. Chen, A.S. Kosky, V.M. Markowitz, and E. Szeto. Constructing and Maintaining Scientific Database Views. In *Proceedings of the 9th Conference on Scientific and Statistical Database Management*, August 1997. 177
- CKMS98. I.A. Chen, A.S. Kosky, V.M. Markowitz, and E. Szeto. Exploring Heterogeneous Biological Databases: Tools and Applications. In *Proceedings of the 6th Conference on Extending Database Technology*, March 1998. <http://www.genelogic.com/opm.htm>. 176, 177, 180, 186
- CM95. I.A. Chen and V.M. Markowitz. An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools. *Information Systems*, 20(5):pp 393–418, 1995. 177, 180
- EST. EST. <http://genome.wustl.edu/est/esthmpg.html>. WashU-Merck Human EST Project. 178
- GMPQ⁺97. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaman, Y. Sagir, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and Languages. *Journal of Intelligent Information Systems*, 1997. (See also <http://www-db.stanford.edu/tsimmis/>). 176
- Lac99. Z. Lacroix. A XML Engine to query Flat File and Web datasources through Object Database Views. Technical report, March 1999. 182
- LEN. LENS. <http://www.agave.humgen.upenn.edu/lens>. Center for Bioinformatics, University of Pennsylvania. 176, 178
- LRO96. A. Levy, A. Rajaraman, and J. Ordille. The World Wide Web as a Collection of Views: Query Processing in the Information Manifold. In *VIEWS96 – Workshop on Materialized Views (in cooperation with SIGMOD 1996)*, 1996. 177
- SAB⁺99. V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J.J. Lu, A. Rajput, T.J. Rogers, R. Ross, and C. Ward. Hermes: A Heterogeneous Reasoning and Mediator System. Submitted to publication – See also <http://www.cs.umd.edu/projects/hermes/>, 1999. 176
- SGD. SGD. <http://genome-www.stanford.edu/saccharomyces/>. Department of Genetics, Stanford University. 176, 183

Understanding and Modeling Business Processes with DEMO

Jan L.G. Dietz

Delft University of Technology, Department ISSE, P.O. Box 356
NL-2600 AJ Delft
j.l.g.dietz@its.tudelft.nl

Abstract. DEMO is a methodology for modeling, (re)designing and (re)engineering organizations. Its theoretical basis draws on three scientific sources of inspiration: Habermas' Communicative Action Theory, Stamper's Semiotic Ladder, and Bunge's Ontology. The core notion is the OER-transaction, which is a recurrent pattern of communication and action. This notion serves as the universal (atomic) building block of business processes. Some fifty projects have been carried out with DEMO, of very different kinds and in various organizations. The success factor has been the same for all these projects, namely the practical relevance of the concepts of DEMO, as well as their clear and precise definitions. The application of the methodology is illustrated taking two well known cases.

1 Introduction

DEMO (Dynamic Essential Modeling of Organizations) incorporates a way of thinking about organization and technology that has originated from a deep dissatisfaction with current ways of thinking about information systems and business processes. These current ways of thinking fail to explain coherently and precisely how organization and ICT (Information and Communication Technology) are interrelated. They fail to provide assistance in articulating what is essential and invariant about the business processes and what are more or less incidental ways of doing. This is however what seems to be needed: separating 'essence' from technology'.

DEMO fits in a fairly new and promising perspective on business processes and information systems, called the Language/Action Perspective, or L/A Perspective for short. The theoretical foundation of this new perspective is constituted by Speech Acts Theory [1], [2], and the Theory of Communicative Action [3]. The pioneer of the L/A Perspective is undoubtedly Fernando Flores [4]. The L/A Perspective assumes that communication is a kind of action in that it creates commitments between the communicating parties. To communicate then is to perform language acts [2] or communicative acts [3], like requesting or promising. Other approaches in the same L/A Perspective can be found in e.g. [5], [6] and [7]. Three international workshops have been held up to now focussing on the L/A Perspective [8], [9], [10]. These proceedings contain several papers concerning DEMO. Some other relevant papers are [11], [12], and [13].

The major difference between DEMO and other L/A approaches, like SAMPO [6], BAT [14], and Action Workflow [7], is that it is built on two additional theoretical pillars next to the Communicative Action Paradigm, namely Stamper's Semiotic Ladder [15] and Bunge's Ontology [16], [17]. The outline of the paper is as follows. Sections 2 through 5 deal with the theory of DEMO. In section 2 the concept of communication is discussed. This constitutes the ground on which the three other core

concepts, information, action and organization are founded. They are discussed in sections 3, 4 and 5 respectively. Section 6 shows how organizations are modeled with the DEMO methodology, and section 7 contains some conclusions.

2 Communication

In line with the overall goal of DEMO to separate essence from technology, we define *communication* as the sharing of thoughts between social individuals or *subjects*. By thought we mean not only pieces of knowledge but also wishes, promises, feelings etc. The unit of communication consists of the sharing of one thought between two subjects, and is called the *communicative action*. The subject that is going to share one of his/her thoughts is called the locutor (L) of the action, and the subject with whom the thought is shared is called the addressee (A). The thought to be shared is formulated in some language that is common to L and A. The produced language expression is called (an elementary piece of) *information*. This information must somehow be made perceivable to A such that it can be interpreted. The effect of interpretation is the creation of a thought of A (of which both L and A usually hope that it is very similar to the thought of L).

Performing a communicative action successfully is not trivial or unproblematic. At least two requirements have to be met. The first one is that there exists a communication channel, i.e. a means by which the language utterances produced by L are transmitted to A. This requirement includes the verification of the identity of the participants. The second requirement is that A understands what L means. Otherwise said, the participants have to verify that the semantic interpretation of the information by L is correct. Assuming that these requirements are fulfilled, we will now focus on the social or intersubjective effect that the performance of a communicative action brings about. As the L/A Perspective tells us, a communicative action consists of an illocutionary action and a propositional action. In the propositional action, a fact in some world is referred to, as well as a time. In the illocutionary action the locutor expresses his/her 'attitude' towards the proposition. In DEMO, the next so-called OER-notation¹ is used to denote communicative actions:

<locutor> : <illocution> : <addressee> : <fact> : <time>

As an example of a communicative action, let us assume that someone, a hotel guest (G for short), addresses a reception employee (E) and utters the next sentence:

"Do you have suites?"

The illocution of the formulated thought is the question, the fact is 'the hotel does have suites', and the time is a not explicitly specified (default) period, most probably 'currently'. The OER-notation of the example communicative action above is:

G : question : E : *the hotel does have suites* : currently

¹ The word "OER" is a Dutch word, meaning "primal", "original". It expresses that one seeks for the essence by abstracting from (current) realization.

The reply by the hotel employee to this question could be:

“Yes, we do.”

The OER-notation of this communicative action is:

E : assertion : G : *the hotel does have suites* : currently

DEMO distinguishes six illocutionary kinds: question, assertion, request, promise, statement and acceptance.

A sequence of to and fro communicative actions between two subjects is called a *conversation*. DEMO distinguishes between informative and performative conversations. *Informative conversations* are conversations in which only questions and assertions occur. An example of an informative conversation between G and E is the combination of the two utterances, mentioned earlier:

G : Do you have suites?

E : Yes, we do

Performative conversations are conversations in which only requests, promises, statements and acceptances occur. Two subtypes are distinguished: actagenic and factagenic. An actagenic conversation is a conversation in which the request and the promise are the chief illocutionary kinds. An example in the hotel situation is:

G : I'd like to have a suite for 3 nights starting January the 3rd

E : Let me see ... yes, I will arrange that for you

The OER-notation of this conversation (in which ‘asap’ means ‘as soon as possible’) is:

G : request : E : *a suite is reserved for G from January 3 till January 6* : asap

E : promise : G : *a suite is reserved for G from January 3 till January 6* : asap

The result of this actagenic conversation is that E has committed him/her-self to make the agreed upon reservation. In a factagenic conversation, the two actors agree on the achieved result. An example of a corresponding factagenic conversation is:

E : Madame, I have reserved a suite for you for 3 nights starting January the 3rd

G : Thank you very much

The OER-notation of this conversation is:

E : statement : G : *a suite is reserved for G from January 3 till January 6* : asap

G : acceptance : E : *a suite is reserved for G from January 3 till January 6* : asap

The example conversations show that the real meaning of a sentence can often not be deduced from a grammatical analysis of the sentence, because it depends heavily on the context (of other sentences) in which it is uttered. Furthermore, the examples show clearly that every sentence possesses the components illocution and proposition (consisting of fact and time).

3 Information

As was already mentioned in the previous section, *information* is understood to be form given thought. Thus, the notion of information is closely related to the notion of communication. To be more specific: there is no information without communication, since information is produced only for the purpose of communicating. Information serves to bridge distances in both (physical) space and time. It allows therefore also communicating with oneself at different points in time. There are three aspects of information to be distinguished, called the *forma*, the *in-forma* and the *per-forma*. These aspects are slight modifications of the semiotic distinctions as proposed in [15].

Every piece of information has a *forma*, meaning that it has some perceivable structure carried in some physical substance. This structure must be recognizable as being an expression in some language, which is the case if the *forma* con-forms to the syntactical rules of that language. The notion of language has to be taken broadly. Every phenomenon that counts as a *forma* according to the institutional rules of a society, is by definition a *forma* (cf. [18]). To every *forma* belongs at least one *in-forma* (if there are more *in-forma*s, these are called homonyms). The *in-forma* is the meaning of the *forma*, the reference to some Universe of Discourse, as defined by the semantics of the language. The aspect *in-forma* also includes the pragmatic rules of the language, like the choice of the right or best *forma* to express some *in-forma* in specific circumstances.

The *per-forma* of a piece of information is the effect on the relationship between the communicating subjects, caused by communicating the thought. It is determined by both the illocution and the proposition of the communicative action, and is further dependent on the current norms and values in the shared culture of the communicating subjects. For example, if one asks someone the time, one is supposed to need the answer (it is not considered just to ask such a question for fun), and the one who answers is expected to answer to the best of his knowledge. However, this norm may be overwritten, e.g. if the one has just robbed the other one. In such a situation, it is e.g. acceptable to be dishonest.

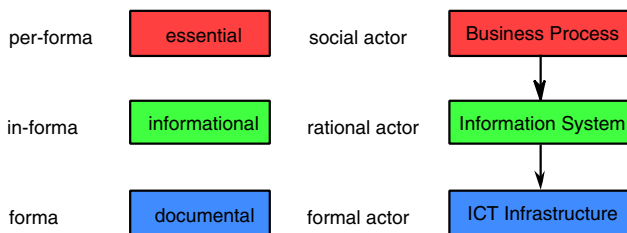


Figure 1 Information aspects, levels, actor roles, and system categories

The distinction between the three aspects *per-forma*, *in-forma*, and *forma*, gives rise to the distinction of three corresponding levels at which information in an organization has to be managed: the essential level, the informational level, and the documental level respectively (cf. figure 1). Likewise, these levels can be understood as 'glasses' through which one can look at an organization. Looking through the *essential* glasses, one observes social actors, i.e. elements that execute per-formative

actions like requesting or promising objective actions, and like stating or accepting results of objective actions (Note. The notion of objective action will be explained in section 4). Looking through the *informational* glasses, one observes rational actors, i.e. elements that execute in-formative actions like collecting, providing, recalling and computing knowledge about per-formative actions and their results. Lastly, looking through the *documental* glasses, one observes formal actors, i.e. elements that execute formative actions like gathering, distributing, storing, copying, and destroying documents containing the aforementioned knowledge about per-formative actions and their results.

Put in other terms, one observes through the three distinct glasses, systems of three distinct categories, viz. business processes, information systems and ICT-infrastructure components (Note. We conform to the habit to speak of business processes instead of business systems). These three system categories and their relationships are also exhibited in figure 1.

The arrows from Business Process to Information System, and from the latter to ICT Infrastructure indicate one-to-many relationships, which should be understood as follows. Given a particular business process, one may conceive of a number of (collections of) information system(s) that support, and in doing so make operational, that business process. However, at any point of time, there is only one (collection of) information system(s). Likewise, given a particular information system, one can conceive of a number of ICT infrastructure configurations that are able to realize, and in doing so make operational, that information system.

4 Action

Action is a core notion in studying any dynamic system from an engineering point of view. According to the distinctions as proposed in the previous section, DEMO distinguishes between essential actions, informational actions and documental actions. The main interest however is in the essential actions, because these are the 'authentic, genuine' business actions, all other actions only serve to support them. The class of essential actions is further divided into objective actions and intersubjective actions.

By executing *objective actions*, the members of an organization fulfill the mission of the organization. The nature of an objective action can be material or immaterial. Examples of material actions are all manufacturing actions in the production of goods as well as all storage and transportation actions. Examples of immaterial actions are the judgement by a court to condemn someone, the decision to grant an insurance claim, and appointing someone president. By executing *intersubjective actions*, subjects enter into and comply with commitments. In doing so, they initiate and coordinate the execution of objective actions. Intersubjective actions are defined to be the performative communicative actions, as presented in section 2, thus requests, promises, statements and acceptances. In order to abstract from the particular subject that performs an action and to concentrate on the functional or organizational role of the subject in performing that action, the notion of *actor* is introduced (e.g. sales and purchasing). Thus, an actor is a particular 'amount' of authority and responsibility; it requires from the fulfilling subject a particular 'amount' of competence. An actor role can be fulfilled by a number of subjects (concurrently as well as collectively), and a subject may fulfill concurrently a number of actor roles.

In correspondence with the distinction between objective and intersubjective actions, DEMO distinguishes between two worlds in which each of these kinds of actions have effect: the *object world* and the *intersubject world* respectively. The effect of every action is a state transition. These transitions are considered to take place instantaneously. A particular transition (e.g. requesting a beer by someone from the barkeeper) at a particular point in time (e.g. the requesting of a beer by a particular person at a particular moment) is called an event. A time series of events is a process.

Objective actions and their related intersubjective actions appear to occur in a particular pattern, called the *OER-transaction*, as illustrated by figure 2. It consists of three phases: the order phase or O-phase, the execution phase or E-phase, and the result phase or R-phase (Note. The three letters O, E and R constitute the Dutch word “OER” which was explained earlier). A transaction is carried through by two actors, who alternately perform actions. The one who starts the transaction and eventually completes it, is called the *initiator* (I in figure 2), the other one, who actually performs the objective action, is called the *executor* (E in figure 2).

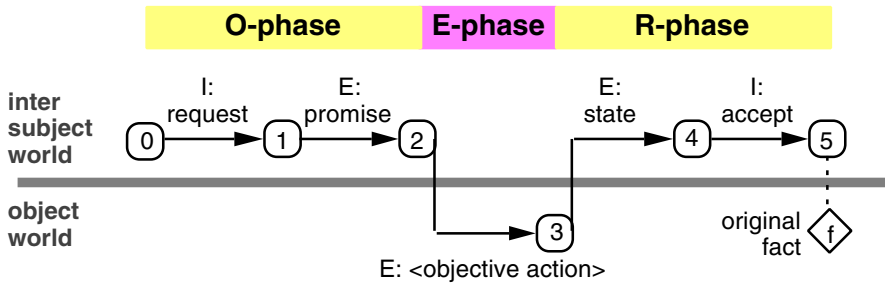


Figure 2 The OER-transaction

The order phase is an actagenic conversation, and the result phase is a factagenic conversation. Both conversations consist of intersubjective actions, having as effect a transition in the intersubject world. These actions are executed alternately by the initiator and the executor of the transaction. In between the two conversations, the objective action is executed, by the executor of the transaction. The effect of this action is a transition in the object world (Note. What figure 2 shows is in fact only the success layer of a transaction process. Next to this layer, a negotiation or discussion layer and a discourse layer are to be distinguished [19]).

Principally, events in the object world are not knowable to the initiator (and to other actors) as long as they are not stated by the executor. This principal position is important. On the one hand, it stresses the supremacy of events in the intersubject world. On the other hand it allows material and immaterial objective actions (and resulting facts) to be dealt with in the same manner. For immaterial facts it is obvious that they cannot be said to exist unless they are stated and subsequently accepted, and thus that they come into existence at the moment of acceptance, i.e. when reaching transaction status 5. Although at first sight, and intuitively, material facts seem to come into existence in status 3, this appears not to be the case on closer observation. In every organization with material objective actions (like manufacturing or transporting firms), there appears always to be someone who is held responsible for a particular material fact being the case. Only he or she has the authority to declare that something is the case, which causes an event in the intersubject world.

The transaction concept as presented serves as the building block of business processes. Otherwise said, every business process is a (molecular) structure of (atomic) transactions. This structure may be arbitrarily complex.

5 Organization

There exists a variety of definitions for the term 'organization'. These definitions represent as many different notions, apparently depending on the point of view one takes towards the phenomenon organization. As indicated in the introduction, DEMO takes an engineering point of view. This is rather new among IS professionals, whose notions of organizations are dominated by the black-box model. Taking the black-box model when studying organizations (or any variant of it like the control model), one tries to find and understand the relevant input variables and output variables, and the relationship between them, called the transfer function. Knowing the transfer function means knowing how the system responds to variations in the values of the input variables. By manipulating the input variables, one is able to change the behavior of the system. If the transfer function is too complicated to understand, the technique of *functional decomposition* can be applied through which the system is replaced by a structure of subsystems of which the transfer functions are more readily understandable.

Like every other system (e.g. an alarm clock or a racing car), the functional behavior of an organization is brought about by the collective working of the constructional components. The construction and the operation of a system are most near to what a system really is, to its ontological description. A very exact and very general, ontological definition of a system is provided by Bunge [17]. Based on this definition, DEMO uses the next definition of an organization:

Something is an *organization* if and only if it fulfills the next properties:

- It has *composition*, i.e. it is composed of actors. These actors act on the basis of assigned authority and with corresponding responsibility.
- It has *effect*, i.e. the actors bring about changes in the object world.
- It has *structure*, i.e. the actors influence each other. Two kinds of mutual influencing are distinguished. *Interaction* consists of executing transactions. *Interstriction* consists of taking into account the results or the status of other transactions when carrying through a transaction.
- The composition is divided into two subsets, called the kernel and the environment, such that every actor in the environment influences, either through interaction or through interstriction, one or more actors in the kernel (and such that there are no isolated parts in the kernel). The closed line that separates the kernel from the environment is called the *boundary*.

Based on this definition, a *white-box model* of a system can be made, as exhibited in figure 3. On the one hand, actors perform objective actions, thus changing the state of the object world. On the other hand, they perform intersubjective actions, thus changing the state of the intersubjective world. When being active the actors take into account (are restricted by) the state of the object world as well as the state of the intersubject world.

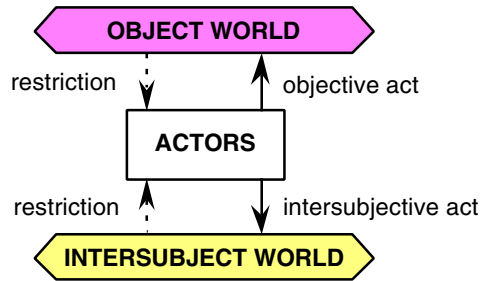


Figure 3 The white-box model

Analogous to the technique of functional (de)composition, there is a technique for composing and decomposing white-box models of a system. It is called *constructional (de)composition*. A central concept in the technique is the concept of subsystem. This concept is in DEMO defined as follows. S2 is a subsystem of S1 if and only if S1 and S2 are systems according to the definition above, the kernel of S2 is a subset of the kernel of S1, the structure of S2 is a subset of the structure of S1, and the environment of S2 is a subset of the composition of S1.

In conclusion, functional (de)composition and constructional (de)composition are similar techniques, but applied to very different system notions. An important consequence is that a functional component does not necessarily have a constructional counterpart. Attempting to relate the one kind of component to the other kind just does not make sense; their natures are inherently incompatible. Therefore, functional decomposition doesn't reveal a bit of the construction and operation of the system.

6 The DEMO methodology

The DEMO methodology was originally intended for carrying out the requirements engineering phase of information systems development, and it is being applied very successfully for that purpose indeed. Next to that however, it appears to be very effective for a variety of other purposes, like Business Process Reengineering, WorkFlow Management, Electronic Commerce and Virtual Organizations. In short, the methodology has proven to be a coherent, integral, and comprehensive approach to the modeling, the (re)design and (re)engineering, as well as the managing, of business processes and information systems. For several years now it is being applied in practice by a growing number of consultancy enterprises in all kinds of organizations: commercial, industrial and governmental.

The main contribution of the methodology in an organization's change process, is the provision of the essential model of the organization. This is a description by means of several diagramming techniques of its business processes (as defined by the DEMO theory), thereby abstracting fully from the informational/documental as well as from the organizational (personnel) realization. In each of the about fifty projects undertaken up to now, it has turned out that this essential model provides an ideal starting point for rethinking firstly the business of the organization as such, and secondly the way in which it is realized now. Invariably, employees and managers are taken by the clarity, the conciseness, the relevance, and the effectiveness of the methodology and the

produced results. Not only do DEMO projects cost a fraction of the costs of a traditional project, it also succeeds where the traditional methodologies keep failing. An extensive discussion of one of the projects is provided in [20].

22 The Ford Case

For the first illustration of the DEMO methodology, we take the well-known Ford Case [21]. Hammer provides the next description of it:

'When Ford's purchasing department wrote a purchase order, it sent a copy to accounts payable. Later, when material control received the goods, it sent a copy of the receiving document to accounts payable. Meanwhile, the vendor sent an invoice to accounts payable. It was up to accounts payable, then, to match the purchase order against the receiving document and the invoice. If they matched, the department issued payment.'

The department spent most of its time on mismatches, instances where the purchase order, receiving document, and invoice disagreed...

One way to improve things might have been to help the accounts payable clerk investigate more efficiently, but a better choice was to prevent the mismatch in the first place. To this end, Ford instituted 'invoiceless processing'. Now when the purchasing department initiates an order, it enters the information into an on-line database. It doesn't send a copy of the purchase order to anyone. When the goods arrive at the receiving dock, the receiving clerk checks the database to see if they correspond to an outstanding purchase order. If so, he or she accepts them and enters the transaction into the computer system. (If receiving can't find a database entry for the received goods, it simply returns the order.)'

According to Hammer, Ford opted by the chosen solution for radical change, and achieved dramatic improvement. To illustrate this, it was mentioned that initially there were 500 people working at the accounts payable department, and that a 75% reduction of this figure was achieved after the solution had been implemented. Let us analyze this case briefly. To start with, figure 4 exhibits (part of) the essential model of the business activities concerned.

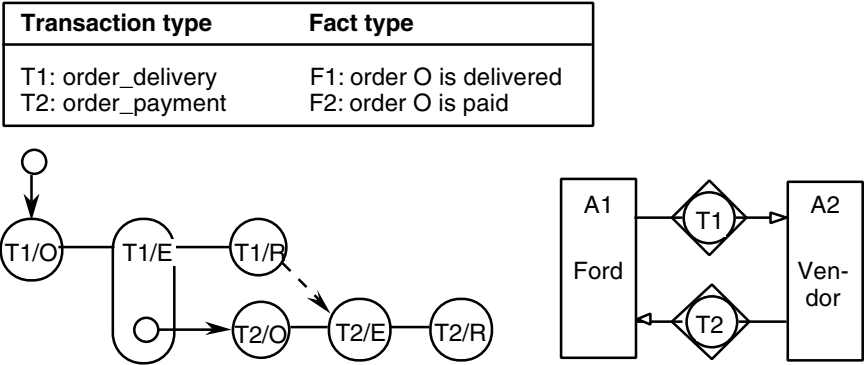


Figure 4 The essential model of the Ford Case

On top of figure 4, a so-called transaction table is shown. It lists the distinct transaction types, including the specification of the resulting fact types. In the Ford Case, there are two transaction types: order delivery, and order payment. In the lower right corner of the figure, the Interaction Model is shown. A box represents an actor. The symbol for a transaction type is a disk with a diamond behind it. The disk represents the intersubject world part and the diamond the object world part. Therefore, the disk may also be interpreted as a communication bank, i.e. the conceptual store of transaction states, and the diamond as a fact bank, i.e. the conceptual store of transaction results and associated facts. A solid line connects the transaction symbol to the initiator of the transaction type, and the solid line with open arrowhead connects it to its executor. For instance, actor A1 is initiator of transaction type T2 and executor of transaction type T1. The lower left corner contains the Process Model. In this diagram, each of the two transaction types is divided into its three constituent phases, the O-phase, the E-phase, and the R-phase. The solid arrows represent initiation relationships. Their meaning is that the process or activity at the point side is started from the process or activity at the shaft side. For every transaction, it holds that the E-phase is preceded by the O-phase, and that the R-phase is preceded by the E-phase. This is shown in figure 4. Also shown is that the O-phase of transaction type T2 is initiated from the E-phase of transaction type T1, and that the O-phase of transaction type T1 is initiated externally (this is indicated by the small circle). The dotted arrow from T1/R to T2/E represents a conditional relationship. It means that T2/E can only be completed after T1/R has been completed. The Process Model in figure 4 expresses that the request for payment is issued in parallel with the statement that the order is delivered, and that the payment is actually performed after the order delivery has been accepted.

Next to the Interaction Model and the Process Model, there are three other model types: the Interstriction Model (describing the inspection relationships between actors and information banks), the Action Model (describing in detail the business rules for carrying through the transactions of the distinct types) and the Fact Model (describing the state space of the object world and the intersubject world). Collectively, these partial models constitute what is called the *essential model* of an organization.

The interesting point in the Ford case is, that the essential model is the same before and after the radical (!) change. In the terminology of DEMO, it means that there has not been a redesign (i.e. a change at the essential level). Redesign would imply that (part of) the essential model, e.g. the Process Model, would have been changed. There has only been a reengineering, i.e. changes are made only at the informational and/or the documental level. In this respect, it sounds misleading to say that Ford has become invoiceless. This is only true at the documental level: there are just no paper invoices sent anymore. The essential meaning of an invoice however is that it is a request for payment, and this request is (of course) still performed, however by different means. The difference between the new situation and the old one is that in the new situation the delivery of goods counts as a request for payment, while in the old situation the paper invoice conveyed this request. All additional information that was contained on the paper invoice in the old situation, like the delivered items, the quantities, and the prices, must now be found elsewhere. In this respect, Ford did make an efficient agreement with its vendors, including that only completely delivered orders would be accepted and consequently paid; the order information was still available at Ford. One could argue that this agreement does affect slightly the essential level (there is a new business rule, stating that only completely delivered orders are accepted). Still, we would not call that a radical change.

6.2 The Elevator Control Case

Since DEMO focuses on the essential level of abstraction, the way in which the essence of a system is realized is hidden. This property offers the possibility to model technical systems as organizations, thus as if they were realized by human actors. Doing this not only makes it possible to reveal the essence of technical systems; it also shows immediately that notions like authority and responsibility apply to these systems as well. On closer view, many technical systems are social systems; the difference lies in the technology used to realize them.

The example taken is well known in the software engineering community, it was an almost classical example in the literature and on conferences in the eighties. A description as well as an analysis can be found in [22]. The description is formulated as the requirements for the design of a program that schedules and controls a set of elevators in a building. Because of the limited space, we do not repeat the complete description here, but confine ourselves to a summary:

The system consists of 4 elevators in a building with 40 floors. Next to the elevator doors at each floor, there is a summons button to go upward and a summons button to go downwards. In the interior of each elevator, there is a panel with 40 destination buttons, for every floor one. Above the door there is also a panel on which is shown at which floor the elevator is arriving (or is stopping). The software program receives summons and destination requests, schedules them, and controls the motors for moving the elevators upwards and downwards and for stopping them at a floor. This should be done in an efficient manner. The program also receives signals from the floor sensors in each elevator shaft, indicating that the elevator approaches the floor. After reception of a summons request, the program sends a signal to illuminate the button. As soon as an elevator visits the floor in the right direction, the light is switched off. A similar procedure holds for the destination buttons in each elevator. The program also sends signals to appropriately illuminate, in the interior of every elevator, the floor, at which it arrives or stops. Lastly, an elevator may be out of service, and a floor may be unreachable, for whatever reason.

A careful investigation at the essential level leads to the identification of four transaction types (cf. figure 5).

transaction type	transaction result
T1 summons_visit	F1 an elevator visits floor F, with direction D
T2 destination_visit	F2 elevator E visits floor F
T3 elevator_movement	F3 elevator E moves in direction M
T4 floor_visit	F4 elevator E visits floor F, with direction D

Figure 5 Transaction-result table of the elevator system

By way of illustration, we will explain which (seemingly non-communicative) actions do count as performing the request, promise, statement and acceptance of transaction T1 (cf. figure 2). Pressing a summons button counts as issuing a request for a T1. Having promised a T1, thus having reached state 2 in figure 2, is represented by the illumination of the button. The opening of the doors and the entering by the passenger of the car can be viewed as the statement and the acceptance respectively, but one might as well say that these actions are implicit. Such implicit actions are as important as the explicit ones, i.e. they should be identified as carefully as the other ones. In case of a breakdown of a transaction, there must be no doubt about which actor is failing to comply with its role, its commitments. Otherwise said, it must always be clear who is responsible.

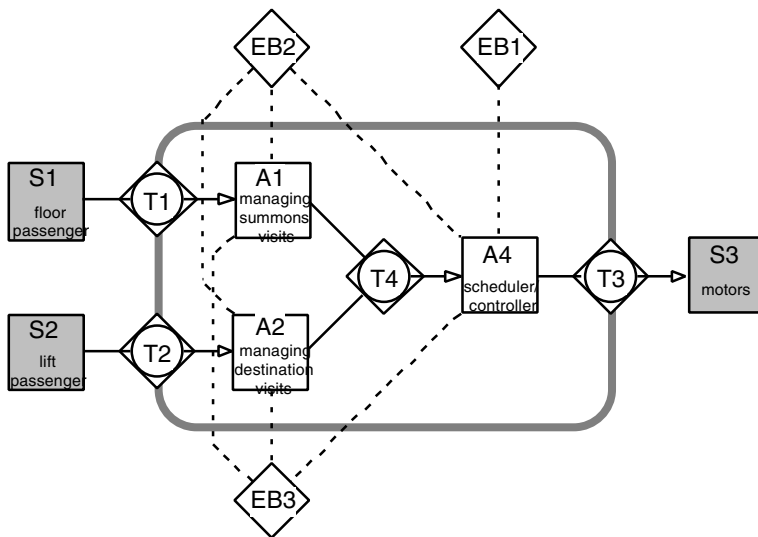


Figure 6 Interaction and Interstriction Model of the elevator system

Based on the table of figure 5, the diagram in figure 6 can be drawn. This diagram exhibits both the Interaction Model and the Interstriction Model: it shows the transaction types with their initiator(s) and executor, as well as the informative conversations between actors via information banks. The gray 'roundangle' represents the system boundary. Since one usually does not have sufficient knowledge about the actors in the environment (or does not care), we draw so-called complex actors or system kernels in the environment: S1, S2 and S3 (In fact, what is inside the boundary, is a revelation of the system kernel S0). Both actor A1 (the executor of T1) and A2 (the executor of T2) are initiator of the internal transaction type T4.

The dotted lines represent inspection links between actors and information banks. The three external banks EB1, EB2 and EB3 contain the elevator position information (from the sensors), the available elevators and the reachable floors respectively. For the analysis of the system, we don't care who is providing this information, even less how it is stored or transmitted. It suffices to know, and to indicate in the models, that a particular kind of information is needed, and that it is available for specific actors.

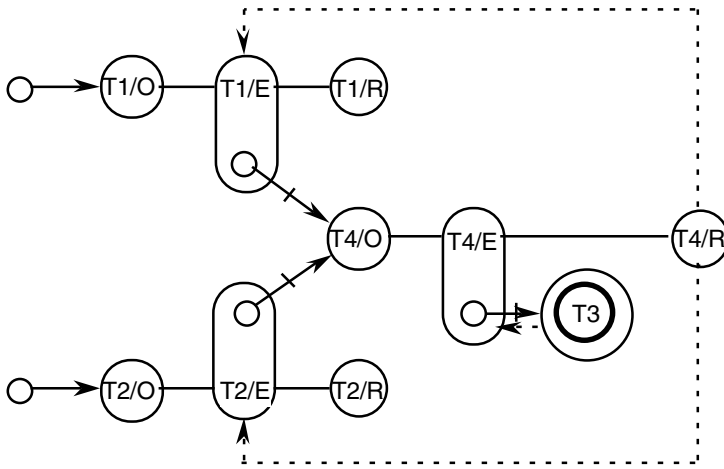


Figure 7 Process Diagram of the elevator system

Lastly, we present in figure 7 the (business) process diagram. It exhibits that T1 and T2 are initiated externally (thus by an actor in the environment), and that a T4 is initiated either during the execution of a T1 or during the execution of a T2. The bars on the initiation links indicate optionality. It means e.g. that not in every instance of T2/E a request for executing a T4 is issued. Practically spoken, this covers the situation that two lift passengers want to go to the same floor. Figure 7 also shows that during the execution of a T4 a set of transactions of type T3 can be initiated. This way of modeling reflects that sometimes there may be no need to issue a motor control command, and sometimes it may be necessary to issue several commands (e.g. to move upwards first and then to stop). Furthermore, the diagram exhibits that the successful completion of a T4 is a condition for the completion of the corresponding T1 and/or T2.

8 Conclusions

The essence of an organization lies in the entering into and the complying with commitments by authorized and responsible subjects. This constitutes the working principle of any organization. The DEMO transaction is the elementary building block of every business process, irrespective of the nature of the business, i.e. of the kind of the objective actions (material or immaterial). At the same time it becomes clear that a business process differs fundamentally from a production or a logistic process, and that so-called information intensive organizations (banks, insurance companies etc.) do have business processes like all other organizations, they only don't have production and logistic processes (at least at the essential level, they do have them at the documental level, like every organization has).

Modeling business processes is a prerequisite for (re)designing and (re)engineering them. Current approaches to modeling business processes however do not embody an appropriate understanding of the notion of business process, and consequently do not provide an effective help. The presented DEMO methodology does offer the appropriate, engineering oriented, understanding. It has been illustrated that the

methodology can be an effective help in various situations concerning the analysis and optimization of business processes. Because one abstracts completely from the way in which the essential model of an organization is realized, one has the right design freedom for thinking anew about that realization. Otherwise said, one is in the right position to think of applying modern information and communication technology in an optimal way.

References

- 1 Austin, J.L., *How to do things with words*, Harvard University Press, Cambridge MA, 1962
- 2 Searle, J.R., *Speech Acts, an Essay in the Philosophy of Language*, Cambridge University Press, Cambridge MA, 1969
- 3 Habermas, J., *Theorie des Kommunikativen Handelns*, Erster Band, Suhrkamp Verlag, Frankfurt am Main, 1981
- 4 Flores, F.M., J.J. Ludlow, Doing and Speaking in the Office, in: *Decision Support Systems: Issues and Challenges*, Pergamon, Oxford, 1980
- 5 Taylor, J.R., *Rethinking the Theory of Organizational Communication: how to read an organization*, Ablex, Norwood, N.J., 1993
- 6 Auramäki, E. E. Lehtinen, K. Lyytinen, A Speech Act based Office Modeling Approach, *ACM TOIS*, vol. 6, no. 2, 1988
- 7 Medina-Mora, R., T. Winograd, R. Flores, F. Flores, The Action Workflow Approach to Workflow Management Technology, *Proc. 4th Int. Conf. on CSCW*, ACM, New York, 1992
- 8 Dignum, F., J.L.G. Dietz, E. Verharen, H. Weigand, *Communication Modeling - the Language/Action Perspective*, Electronic Workshops in Computing, Springer, 1996, <http://www.springer.co.nk/ewic/workshops/CM96/>
- 9 Dignum, F., J.L.G. Dietz, *Proceedings of the Second International Workshop on Communication Modeling*, Computing Science Reports, Eindhoven University of Technology, 1997
- 10 Goldkuhl, G., M. Lind, U. Seigerroth, *Proceedings of the Third International Workshop on Communication Modeling*, Department of Informatics, Jönköping International Business School, 1998
- 11 Dietz, J.L.G., Modeling Business Processes for the Purpose of Redesign, in: *Proc. IFIP TC8 Open Conference on BPR*, North-Holland, Amsterdam, 1994
- 12 Van der Rijst, N.B.J., V.E. van Reijswoud, Comparing Speech Act Based Modeling Approaches for the Purpose of Information Systems Development, in: *Proc. of the Third European Conference on Information Systems*, Athens, Greece, 1995
- 13 Dietz, J.L.G., J.B.F. Mulder, Realizing Strategic Reengineering Objectives with DEMO, in: *Proc. International Symposium on Business Process Modeling*, Springer-Verlag 1996
- 14 Lind, M., G. Goldkuhl, Reconstruction of Different Business Processes: A Theory and Method Driven Analysis, in: [Dignum, Dietz, 1997]
- 15 Stamper, R.K., Applied Semiotics, in: *Proc. of the ICL/University of New Castle Seminar Information*, New castle, 1993
- 16 Bunge, M.A., *Treatise on Basic Philosophy*, vol.3, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1977
- 17 Bunge, M.A., *Treatise on Basic Philosophy*, vol.4, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1979
- 18 Searle, J.R., *The Construction of Social Reality*, Allen Lane, The Penguin Press, London, 1995
- 19 Van Reijswoud, V.E., *The Structure of Business Communication - Theory, Model and Application*, Doctoral Thesis, Delft University of Technology, 1996

- 20 Van Reijswoud, V.E., J.B.F. Mulder, J.L.G. Dietz, Speech Act Based Business Process and Information Modeling with DEMO, *Information Systems Journal*, 1999
- 21 Hammer, M., Reengineering Work: Don't Automate, Obliterate', *Harvard Business Review*, 1990
- 22 Yourdon, E., *Modern Structured Analysis*, Prentice-Hall, Inc., 1989

For more information the reader is referred to the website www.demo.tudelft.nl

A Methodology for Building a Repository of Object-Oriented Design Fragments

Tae-Dong Han, Sandeep Purao, and Veda C. Storey

J. Mack Robinson College of Business Administration, Georgia State University,
P.O. Box 4015, Atlanta, Georgia 30302-4015
{than, spurao, vstorey}@gsu.edu

Abstract. Reuse is as an important approach to conceptual object-oriented design. A number of reusable artifacts and methodologies to use these artifacts have been developed that require the designer to select a certain level of granularity and a certain paradigm. This makes retrieval and application of these artifacts difficult and prevents the simultaneous reuse of artifacts at different levels of granularity. A specific kind of artifact, analysis pattern, spans these levels of granularity. Patterns, which represent groups of objects, facilitate further assembly into what we call design fragments. Design fragments can then be used as reusable artifacts in their own right. A methodology for building a repository of design fragments is presented that consists of core and variant design fragments. The effectiveness of the methodology is assessed by verifying the appropriateness of the design fragments generated through a clustering process.

1. Introduction

Reuse involves the design of new systems from prefabricated reusable artifacts [7]. Although a number of reusable artifacts at various granularity levels have been created, they have been less effective than desired. Class libraries and business objects, for example, are difficult to reuse since finding a component that fits into a specific application can be problematic. On the other hand, as the abstraction and granularity of an artifact increases, as found in domain and reference models, a greater degree of modification is required, either to the model or the business process itself. One kind of reusable artifact, analysis patterns [2, 3], represents an opportunity to overcome these problems. Although patterns themselves are fairly small and at a high level of abstraction, they can be used and synthesized to create a larger-grained, specific reusable artifact, which may, in turn, be supplemented with analysis patterns.

Our prior research presented an approach to reuse-based design with analysis patterns [8, 9, 10]. The results suggested that analysis patterns can present many challenges to the designers, such as searching for appropriate patterns, instantiating them in the problem domain, and synthesizing them to create an initial design. In this research, we address these challenges by creating a larger and more specific artifact, that we call a *design fragment*. The analysis patterns provide us with a means of structuring these design fragments in an indexed repository so that they can be easily accessed in new design situations. The objective of this research, therefore, is to:

develop a methodology for building a repository of reusable design fragments from previously developed object-oriented conceptual designs.

We develop a mechanism that creates these design fragments from prior designs and indexes and stores them, so that an automated search can be performed to find the design fragments and possible extensions appropriate for new system requirements. The contribution of the research is to define design fragments, and present an approach to create, categorize, and index them in a repository that will greatly reduce search and modification costs for systems development.

This paper is divided into six sections. Related research is described in section 2. Sections 3 and 4 outline our methodology and clustering algorithms. The results are demonstrated in section 5 by application to three new design situations. Section 6 concludes the paper.

2. Related Research

2.1 Analysis Patterns

An analysis pattern¹ is a group of generic objects, defined in a domain-neutral manner, which can be applied, by analogy, in different domains [2]. Each generic object, such as *Actor* or *Transaction*, has stereotypical properties and responsibilities. Libraries of analysis patterns have been created for reuse [2, 3]. For example, the pattern *Participant* – *Transaction* can be used in different domains to create instantiations, such as ‘cashier – sale’, ‘officer – incident’ or ‘clerk – reservation.’ Figure 1 shows the analysis pattern *Participant* – *Transaction*.

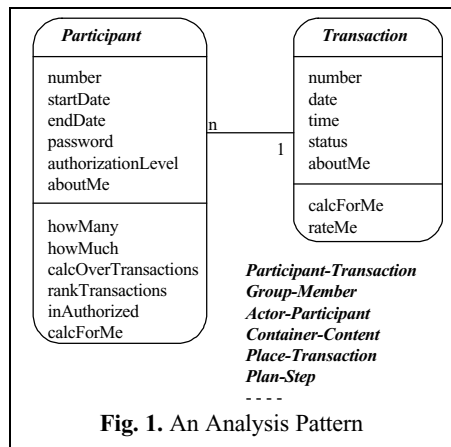


Fig. 1. An Analysis Pattern

2.2 Creating Synthesized Designs with Reuse of Analysis Patterns

Assembling systems with reuse of analysis patterns involves the retrieval, instantiation, and synthesis of patterns in the problem domain. In previous work [8, 9, 10], we have proposed a reuse-based design process to facilitate these tasks. As shown in Figure 2, the naïve approach to synthesizing analysis patterns into a design begins with a simple natural language assertion such as: “a system to track sales at different stores.” Using natural language, instead of creating a formal specification language, frees designers from the burden of learning and commanding a new

¹ Patterns [1] have been identified at the conceptual design stage (analysis patterns [2, 3]) as well as the detailed design stage (design patterns [4]).

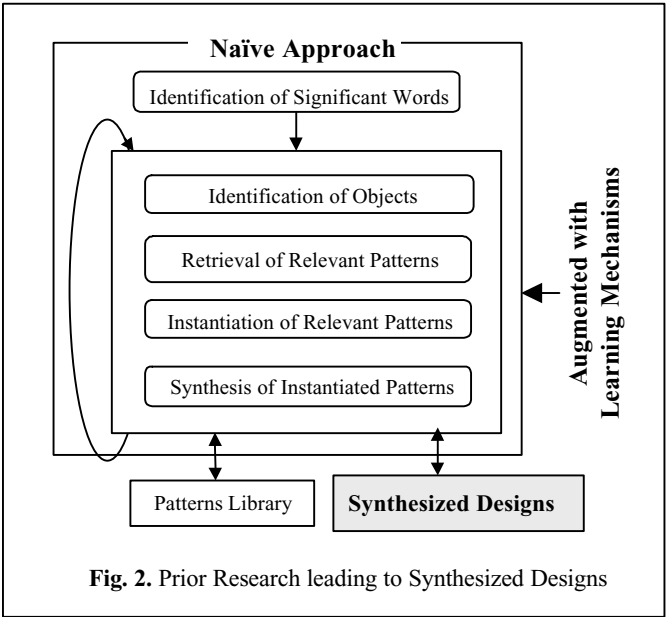


Fig. 2. Prior Research leading to Synthesized Designs

language. These natural language assertions are parsed, retaining words such as 'sale.' Appropriate objects are identified such as 'Transaction' (for sale). Relevant patterns are retrieved, such as *Place – Transaction*. These patterns are instantiated, using keywords identified in the problem assertion, for example, 'store – sale' (for the pattern *Place – Transaction*). Finally, the instantiated patterns

are synthesized using overlapping objects. The process can iterate with designer interaction. We have augmented this naïve approach with learning mechanisms that exploit the Usage History and the structure of patterns (e.g. the participation of an object in multiple patterns), to improve the process and provide greater assistance to the designer. Figure 2 shows this approach. The synthesized designs created from these approaches represent the precursor to the methodology described in this paper. Each design contains multiple analysis patterns instantiated and synthesized to address a specific set of requirements. They form the key inputs to this research. We use clustering to create a new type of reusable artifacts, design fragments.

2.3 Clustering as the Mechanism for Constructing Design Fragments

Clustering classifies a set of objects into sets that share common features. It involves unsupervised, inductive learning in which “natural classes” are found from elements of a set [5]. Clustering is based on close associations or shared characteristics, that is,

a *similarity measure*. Clusters represent collections of elements whose intra-cluster similarity is high and inter-cluster similarity is low. In statistical clustering, this similarity can be decided by a numerical distance. However, numerical clustering cannot take into account the *meaning* of the elements or the clusters. Clustering can be conceptual using semantic and syntactic knowledge about

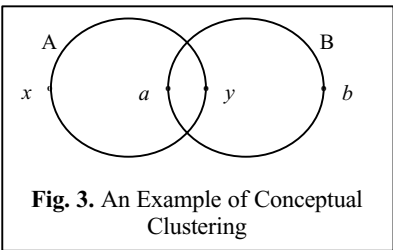


Fig. 3. An Example of Conceptual Clustering

the elements. For example, in Figure 3, the points *a* and *y* are grouped into the one cluster based on numerical methods. However, with knowledge of the circle, it is

more appropriate to cluster x and y into group A, and a and b into group B [6]. We use mechanisms analogous to these to construct the design fragments.

3. Building Design Fragments

This section presents the methodology for constructing reusable design fragments as identified by clustering algorithms. The clustering is based on: similarities in requirements statements and similarities in the composition and ontological classifications of elements in the synthesized designs.

3.1 Design Fragments

A design fragment is a specific instantiation and arrangement of analysis patterns that addresses a specific set of requirements. It is a partial design, that is, a group of patterns that are instantiated and synthesized to address a specific set of requirements. Figure 6 shows sample design fragments. A design fragment embodies higher granularity than an analysis pattern (Figure 4). It provides the same level of specificity as a domain model since it is a partial design.

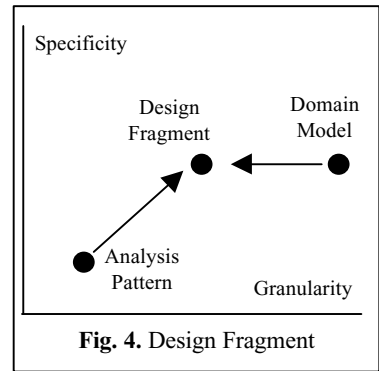


Fig. 4. Design Fragment

3.2 Solution Architecture

Figure 5 shows our approach to building the design fragments. The area enclosed by the dotted line indicates our prior research, which generates conceptual designs by synthesizing analysis patterns. This research focuses on creating and classifying design fragments from designs assembled using our prior research. The classification proceeds as an application of successive clustering algorithms that exploit the: (a) problem description, (b) design graph, and (c) ontological classification of pattern instantiations. The resulting groups of instantiated patterns represent different partial designs that we call design fragments. These are indexed and stored in a repository. The methodology is applied to designs “within” a domain since the objective is to create design fragments that are more “specific” than analysis patterns.

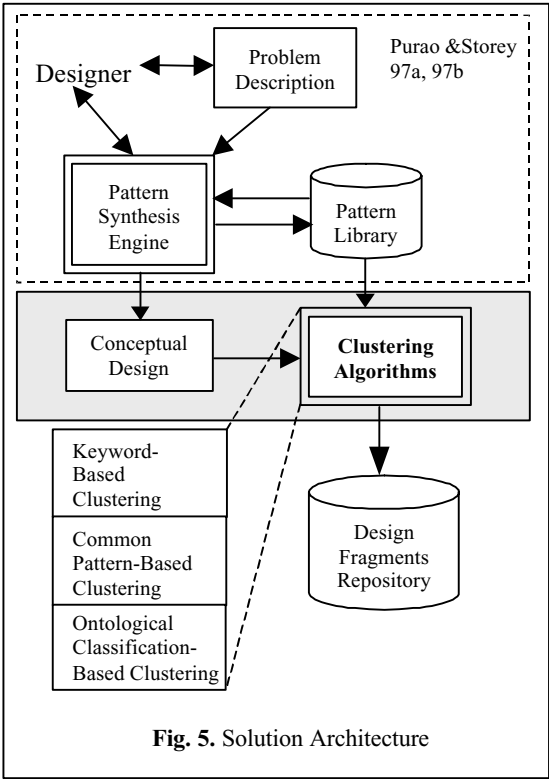


Fig. 5. Solution Architecture

3.3 A Methodology for Building Design Fragments

This section illustrates the methodology by demonstrating the creation and indexing of design fragments from ten designs in the human resource management domain. Figures 6 and 7 show (a) several designs in this domain generated by synthesizing analysis patterns, and (b) important keywords from the initial problem description documents identified using natural language parsing techniques. Although these ten designs are similar, only three (1, 7, and 8) are identical. Designers often need to create different designs for similar applications as dictated by their specific requirements. Each design represents instantiations and synthesis of a different set of patterns. Design 1 (Figure 6),

for example, represents a synthesis of six patterns: *Actor – Participant* (company – employee), *Participant – Transaction* (employee – agreement), *Place – Transaction* (department – agreement), *Transaction – Subsequent Transaction* (agreement – subsequent agreement), *Transaction – Transaction Line Item* (agreement – agreement line item), and *Transaction – Specific Item* (agreement – specific item). The complete set of ten designs (from Figures 6 and 7) represent the input to our example.

These ten designs undergo successive clustering algorithms to identify, index, and cluster the design fragments. The output is a repository that stores classified design fragments based on three criteria: important keywords, common pattern sets, and the ontological classification of instantiations.

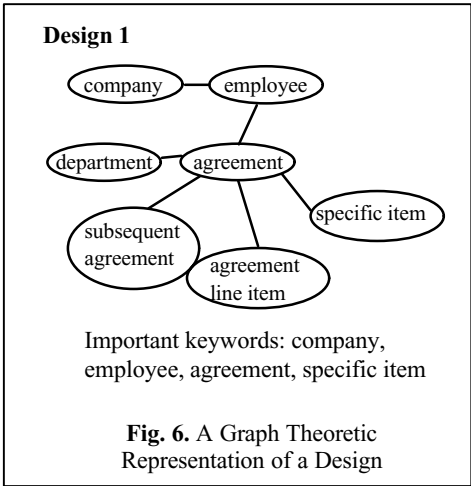


Fig. 6. A Graph Theoretic Representation of a Design

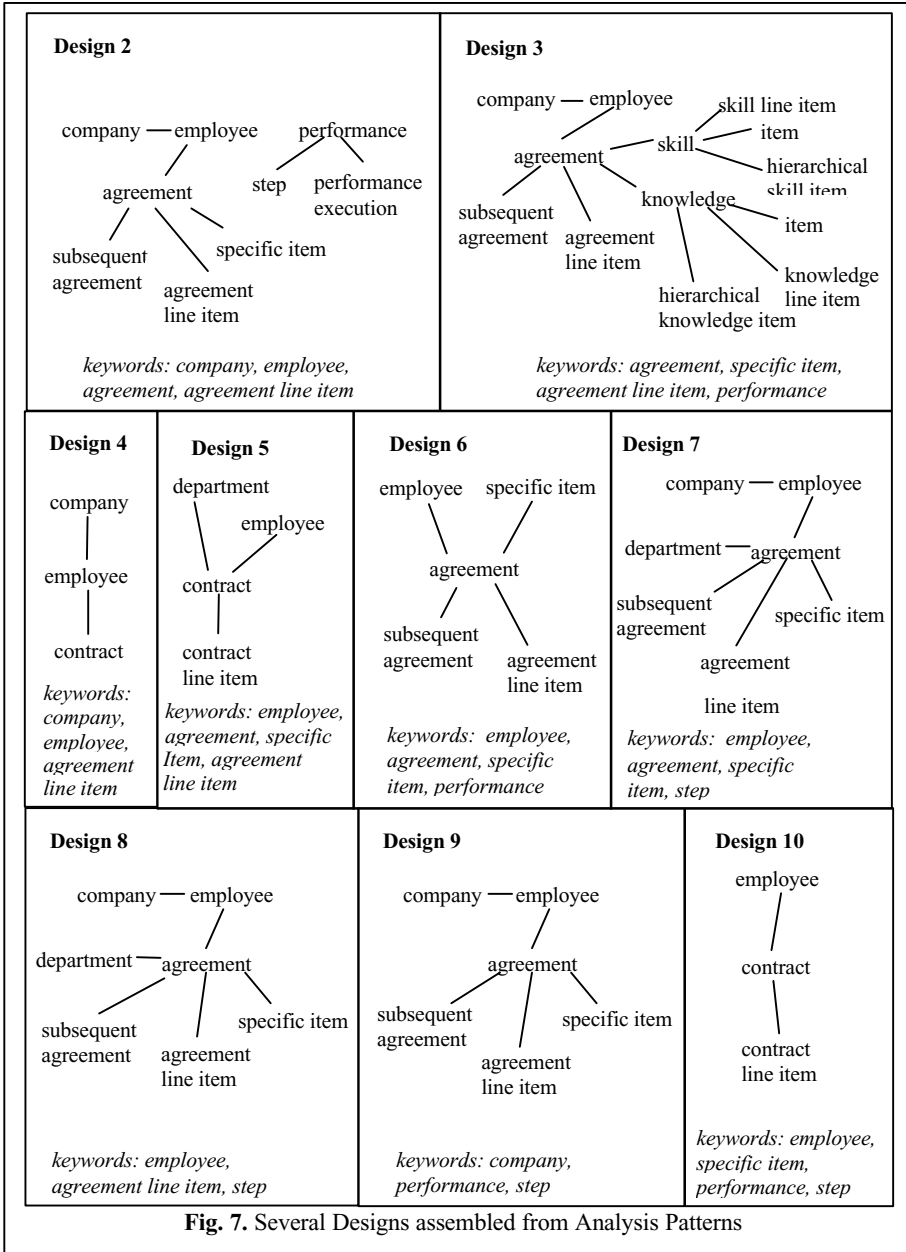


Fig. 7. Several Designs assembled from Analysis Patterns

3.3.1 Creating Keyword-Based Clusters

The designs are classified using keywords ($k \in K$) in the requirements statements. Each set of keywords, that is, a unique requirement, is denoted as R_i and defined by important keywords; $R_i(k_1, k_2, \dots)$. We denote each cluster as a unique set of requirements (R_1, R_2 , etc). Each design is classified into one or more clusters

depending upon the elements (keywords) that make up its requirements. The clustering algorithm interprets the relative frequencies of keyword pairs as measures of similarity. Table 1 shows the sample designs placed into nine clusters. A given design may be classified into one/more clusters or may even form a cluster of one. Design 1 (D_1) is classified into cluster R_1 along with designs 5, 6, and 7, and into cluster R_5 with design 2. Design 9 forms a cluster of one.

Table 1. Design clusters based on keywords

Keyword-based clusters	Designs in the cluster
R_1 (employee, agreement, specific item)	D_1, D_5, D_6, D_7
R_2 (company, employee, agreement line item)	D_2, D_4
R_3 (employee, agreement, agreement line item)	D_2, D_5
R_4 (employee, agreement line item)	D_4, D_8
R_5 (company, employee, agreement)	D_1, D_2
R_6 (employee, specific item, step)	D_7, D_{10}
R_7 (agreement, specific item, agreement line item)	D_3, D_5
R_8 (agreement, specific item, performance)	D_3, D_6
R_9 (employee, specific item, performance)	D_6, D_{10}
R_{10} – Remainder	D_9

3.3.2 Creating Common Pattern Set-Based Clusters

The designs are also clustered based on commonalities in patterns used. A Common Pattern Set (CPSn) is the maximal set of patterns shared among the designs. For example, if some designs share only the patterns *Actor – Participant – Transaction*, that is, the composition of two analysis patterns, *Actor – Participant* and *Participant – Transaction*, this represents the CPS for those designs. CPS's represent invariant occurrences of design fragments among different designs. They allow clustering of designs and identification of first-cut design fragments. Table 2 shows that five CPS's (that is, design fragments) are identified from ten sample designs. Design 1 (D_1) is clustered into the CPS5 group with designs 7 and 8. Figure 8 shows the graphical representation of each CPS. The clusters represent cumulative design fragments, that is, the relationships between clusters represent aggregation. It is possible that the clustering process may diverge into multiple branches, leading to design fragments that do not obey a linear aggregation hierarchy.

Table 2. CPS-based Clusters

CPS-based design fragments	Designs in the cluster
CPS1: Participant – Transaction	D_4
CPS2: Participant – Transaction, Transaction – Transaction Line Item	D_5, D_{10}
CPS3: Participant – Transaction, Transaction – Subsequent Transaction, Transaction – Transaction Line Item, Transaction – Specific Item	D_6
CPS4: Actor – Participant, Participant – Transaction, Transaction – Subsequent Transaction, Transaction – Transaction Line Item,	D_2, D_3, D_9

Transaction – Specific Item	
CPS5: Actor – Participant, Participant – Transaction, Place – Transaction, Transaction – Subsequent Transaction, Transaction – Transaction Line Item, Transaction – Specific Item	D ₁ , D ₇ , D ₈

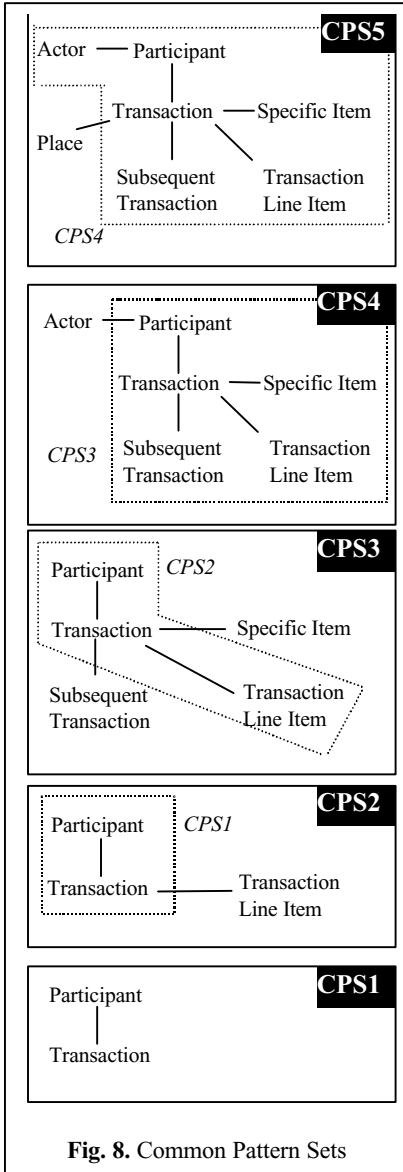


Fig. 8. Common Pattern Sets

Although the CPS's are cumulative, the membership of designs in the clusters indicated by these CPS's is not. This is because a CPS is designated as the *maximal* set of common patterns. For instance, CPS2 is fully contained in CPS3; that is, design 6 (D₆) also contains CPS2. However, D₆ is a member of the cluster defined by CPS3, but not CPS2. The clustering algorithm works by identifying the maximal set of common patterns at each iteration.

3.3.3 Refining the Clusters using Ontological Classification of Instantiations

The final result requires one more level of classification (not demonstrated here due to space limitations). Each CPS is refined according to how each object in the patterns is instantiated. One way to understand the semantics of an object is to map it to an ontology. We adopt the ontology of Storey et al. [11, 12] and store ontological classifications for each object in a pattern. For example, instantiations of *Participant* such as 'agent', 'applicant', 'cashier', or 'clerk' are ontologically categorized as *person*. Instantiations such as 'manufacturer', 'supplier', or 'shipper' have an ontological classification of *tradable social structure*. Similarly, *Transaction* can be instantiated into two ontological types, *event* and *activity*. Thus, for a pattern *Participant – Transaction*, four ontological groups exist: *person – event*, *person – activity*, *tradable social structure – event*, and *tradable social structure – activity*.

The number of types of ontological relationships for different CPS's varies according to how many ontological types are

available for each pattern within the CPS. More precisely, a CPS is presented as CPS_{n_j}, where *j* indicates the type of the ontological group. In CPS5, for example, both *Actor* and *Participant* can be instantiated as *person* (i.e., person, agent) or *social structure* (i.e., company, supplier). *Transaction* can be an *event* (i.e., withdrawal), or

an *activity* (i.e., delivery). *Place* can be a *tradable stationary place* (i.e., bank), a *non-tradable stationary place* (i.e., region), or a *non-stationary place* (i.e., shelf). Thus, there may be numerous instantiations of CPS5, and may need refinement accordingly.

4. Clustering Algorithms

The clustering mechanisms derive and index the design fragments using similarity measures in: (a) requirements, as captured by keywords, and (b) the patterns, their instantiations, and arrangements. Two clustering mechanisms are developed: 1) top-down, exploiting keywords, and 2) bottom-up, exploiting patterns for clustering.

4.1 Top-Down Algorithm

Step1: Identify Keywords. The requirements statements for each prior design can be parsed to identify candidates for keywords. From these, a subset is identified as the ‘keywords’ for the domain. The candidate words (k_i), with a higher relative frequency (kf_i), represent the keywords in the domain. The relative frequency is the number of occurrences of a keyword (nk_i) in all problem statements (nd) i.e. keyword relative frequency: $kf_i = nk_i / nd$. A threshold may be used to determine the important keywords in the domain. For our example, we identify keywords from the ten examples: k_1 : company, k_2 : employee, k_3 : agreement, k_4 : specific item, k_5 : agreement line item, k_6 : performance, k_7 : step.

Step 2: Define each Requirements Statements as a Set of Keywords. Each design is represented as a set of important keywords, $D_n[\{k_i\}]$, for example, $D_{50}[k_1, k_3, k_5, k_9]$. The designs are shown in Table 3. (See Figures 6 and 7.)

Table 3. Important keywords identified in each design

$D_1[k_1, k_2, k_3, k_4]$	$D_2[k_1, k_2, k_3, k_5]$	$D_3[k_3, k_4, k_5, k_6]$	$D_4[k_1, k_2, k_5]$	$D_5[k_2, k_3, k_4, k_5]$
$D_6[k_2, k_3, k_4, k_6]$	$D_7[k_2, k_3, k_4, k_7]$	$D_8[k_2, k_5, k_7]$	$D_9[k_1, k_6, k_7]$	$D_{10}[k_2, k_4, k_6, k_7]$

Step 3: Compute Similarity Measure. The similarity measure that clusters the designs is keyword cohesion, which indicates the degree of common keywords among requirements statements. However, computing cohesion of all possible combinations of keywords is an NP-complete problem. Thus, keyword cohesion, kc_{ij} , is computed only between every pair of keywords and exploited during clustering. The computation consists of $kc_{ij} = nk_{ik_j} / nd$ where nk_{ik_j} is the number of occurrences of keywords k_i and k_j together in the requirements statements, and nd is the number of designs. The keyword cohesion table is shown in Table 4.

Table 4. Keyword Cohesion

	k_1	k_2	k_3	k_4	k_5	k_6	k_7
k_1		3	2	1	2	1	1
k_2			5	5	4	2	3
k_3				5	3	2	1
k_4					2	3	2
k_5						1	1
k_6							2
k_7							

Step 4: Cluster Designs using the Similarity Measures. The clustering process begins with keyword pairs that appear most often in documents. $[k_2, k_3]$, $[k_2, k_4]$, and $[k_3, k_4]$, have the highest cohesion rate, $kc_{23} = kc_{24} = kc_{34} = 5/10 = 50\%$. Their union $[k_2, k_3, k_4]$ is first checked for coexistence and coverage in the requirements statements (Table 3). The requirements statements for four designs (D_1 , D_5 , D_6 , and D_7) share all three keywords. These keywords cover 75% of the keywords in each design. We interpret this as these four documents share a common ‘concept,’ denoted as C_i (same as R_i in Table 1). A threshold may be used to ensure that the current ‘concept’ covers a significant fraction of the requirements statement. For this example, we use a threshold of 60%². Since the threshold is met by all designs, they are placed in a cluster. The next highest cohesion rate is kc_{25} ($4/10 = 40\%$). The requirements statements for four designs (D_2 , D_4 , D_5 , and D_8) share these keywords $[k_2, k_5]$. However, they cover only 50%, 67%, 50%, and 67% (respectively) of the requirements. Since the threshold (60%) is not met for all designs, the next step is to find a common set of keywords (a ‘concept’) in all possible combinations of these designs, that satisfies the threshold. As a result, three sets of keywords are identified that are shared by different combinations of designs. $[k_1, k_2, k_5]$ are shared by two designs (D_2 and D_4) at 75% coverage level respectively. $[k_2, k_3, k_5]$ are also shared by two designs (D_2 and D_5) at 75% coverage level. And $[k_2, k_5]$ are shared by D_4 and D_8 at 67% coverage level. Thus, D_2 and D_4 are clustered into $C_2(k_1, k_2, k_5)$. D_2 and D_5 are clustered into $C_3(k_2, k_3, k_5)$, and D_4 and D_8 into $C_4(k_2, k_5)$. By repeating this process for each next successive cohesion level in Table 4, nine concepts are identified (Table 1). This algorithm is:

The Top-Down Algorithm

1. Find pairs of keywords with the next highest cohesion frequency.
2. Find the union of keyword from these pairs.
3. Identify all req’t statements where keywords in this keyword union exist, ‘n’.
4. Compute req’t_coverage, i.e. ‘keywords in keyword union’ / ‘keywords in req’t statements’ for all these designs.
5. Cluster all designs only when req’t_coverage $\geq$ threshold is true for all designs. If step 5 cannot be satisfied then continue to step 6, otherwise return to step 1.
6. Find maximal subset of keywords in all possible combination of (n-1) requirements statements.
7. Compute req’t_coverage for each combination of designs.
8. Cluster all designs only when req’t_coverage $\geq$ threshold is true for each combination of designs. If any more designs remain, set n to n-1 and return to step 6, else return to step 1, until $n-1 \geq 2$.

4.2 Bottom-up Algorithm

The bottom-up clustering algorithm is composed of two steps; first, identifying common pattern sets (CPS’s) as features that cluster designs, and second, further discriminating each cluster of designs (based on CPS) with its ontological meaning.

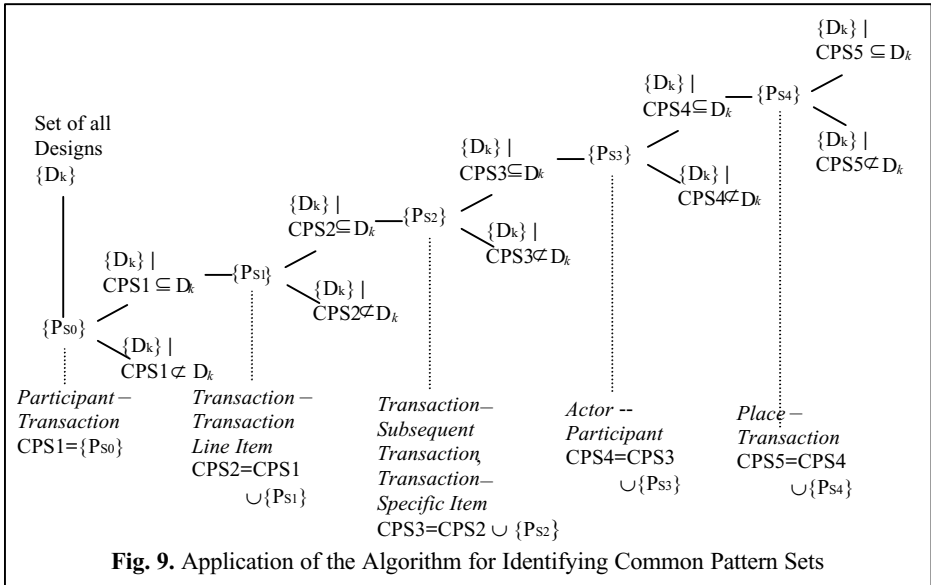
² When none or few designs meet the designer-determined threshold, it means that few requirements statements share common concepts, that is, should be clustered separately.

Step 1: Identify Common Pattern Sets. Finding CPS's among designs begins with a seed pattern set (P_s); that is, the most frequently used pattern(s) in all designs. Continuing the example, pattern Participant – Transaction represents the seed pattern. The designs are divided into two groups – one that includes this pattern and another that does not. Here, all ten designs include the seed pattern. Within each group of designs, the next most frequently shared pattern set is identified, and the grouping repeated. The Transaction – Transaction Line Item pattern meets this criterion in the group that has the seed pattern. It is shared by nine designs (all except design 4 (D_4)). The process continues until there are no more shared patterns. Application of the algorithm is shown in Figure 9. The algorithm can be used with a threshold (e.g. most common pattern must be shared by >50% in the group). This algorithm is:

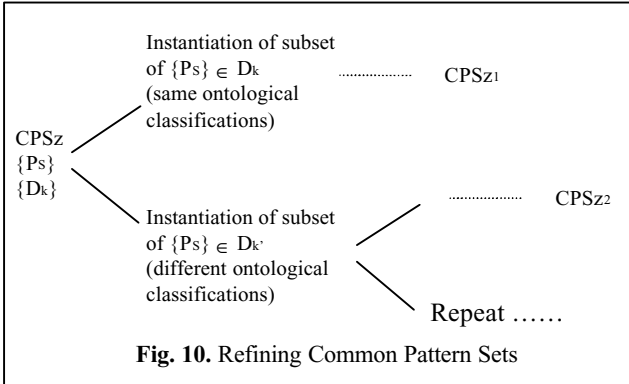
The Bottom-up Algorithm

1. Find the next seed pattern(s) (within threshold) in the *current set of designs*.
2. Name this set of seed pattern(s) as common pattern set (CPS_i)
3. Create a subset of designs where $CPS_i \subseteq D_k$ and another subset where $CPS_i \not\subseteq D_k$
4. Redefine the subset with $CPS_i \subseteq D_k$ as the *current set of designs*.
5. Repeat 1 thorough 4 until no further CPS is found.

The result of this process is the membership of different designs in CPS's. For example, designs D_1 , D_7 , and D_8 share CPS5, whereas designs D_2 , D_3 , and D_9 share CPS4. Table 2 shows the correspondence between CPS's and different designs.



Step 2: Discriminate CPS's using ontological classification of instantiations. The CPS's provide an approximation of the shared design elements – analysis patterns – in the different designs. However, the same pattern may be instantiated in very different ways in different designs. For example, *Participant – Transaction*,



instantiated as ‘cashier’ for *Participant* and ‘session’ for *Transaction* in one design (D_k) and as ‘supplier’ for *Participant* and ‘shipment’ for *Transaction* in another design (D_l). In D_k , the instantiation ‘cashier’ suggests the ontological classification, *person*,

whereas the instantiation ‘supplier’ suggests the ontological classification, *tradable social structure*. Although similar patterns are used in the two designs, their specific instantiations are different. By comparing the ontological classification of the instantiations, we can further refine the designs within each CPS group. Consider a common pattern set, CPS_k , in which three designs (say D_1 , D_2 , and D_3) are classified and part of CPS is *Participant – Transaction* pattern. It is instantiated as ‘employee’ (ontological classification *person*) – ‘contract’ (ontological classification *event*) in two designs (say D_1 and D_2); that is, it has the same corresponding ontological classification. The other design, D_3 , has an instantiation ‘employee’ (ontological classification *person*) – ‘session’ (ontological classification *activity*) that is a different ontological classification. As a result, CPS_k is further refined into CPS_{k1} (with D_1 and D_2) and CPS_{k2} (with D_3). The algorithm for this refinement is shown below and an application shown in Figure 10.

Ontological Refinement Algorithm

1. In all designs that share the next lowest CPS, determine the ontological classification of instantiations of all patterns in that CPS (say CPS_z). Let $n = 0$ to track refined CPS’s.
2. Determine the next maximal subset of patterns from the current CPS such that the ontological classifications of their instantiations are equivalent.
3. Create a new $CPS_{z(n+1)}$ consisting of that maximal subset of patterns.
4. Redefine the current CPS as $CPS_{z1} - CPS_{z(n+1)}$. If this is not NULL, return to step 2. Increase n by 1.
5. Repeat by returning to step 1.

5. Assessment of Results

The reusable artifacts, *design fragments*, and the methodology developed provide the set of clustered design fragments shown in Table 2 and Figure 8. The process described above, thus, parses the available designs in a domain to create design fragments that may be (a) accessed, (b) suggested, (c) extended, and (d) refined. Keyword-based clustering provides (a) an *access* mechanism to design fragments. Common-pattern-set-based clustering (b) provides *suggestions* for design fragments

and (d) *extends* suggestions to alternative designs. Finally, the ontological-classification-based clusters can be used to (c) *refine* the chosen design fragments. We illustrate the usefulness of our approach using three cases from the same domain. Table 5 shows the results.

Table 5. Suggested Starting Points for Three New Design Situations

Case	Keywords	'Concepts' identified	Designs corresponding to this 'concept' (Table 1)	Possible starting points for the designer	
				Design fragments (CPS's) where these designs are clustered	Designs in that cluster (Table 2)
Case 1: "To keep track of personal information about employee, to efficiently manage the payroll, benefits and other information related to employee, and to help decision-making step on employee's incentive plan and hiring process."					
1	Employee, step	C ₆	D ₇	CPS5	D ₁
					D ₇
					D ₈
			D ₁₀	CPS2	D ₅
					D ₁₀
Case 2: "To track personal information about company employee, to identify current employee job category, classification, and salary, to track employee evaluations and salary changes, and to track the benefits being used by each employee."					
2	Company, employee	C ₂	D ₂	CPS4	D ₂
					D ₃
					D ₉
			D ₄	CPS1	D ₄
		C ₅	D ₁	CPS5	D ₁
					D ₇
					D ₈
			D ₂	CPS4	D ₂
					D ₃
					D ₉
Case 3: "To improve recruitment contract process, to increase knowledge base about company, and to ensure fast tracking of employee agreement."					
3	Company, employee, agreement	C ₅	D ₁	CPS5	D ₁
					D ₇
					D ₈
			D ₂	CPS4	D ₂
		D ₃			
		D ₉			

For case 1, the designer enters the problem description and, 'employee' (k₂) and 'step' (k₇) are identified as important keywords (Refer to Step 1 in section 4.1). From

these, the common keyword concept (C_6) is identified and designs retrieved that are clustered in the concept. Two designs (D_7 and D_{10}) are retrieved from that cluster. However, they have quite different shapes, that is, they are classified into two different design groups by CPS criterion. Since D_7 belongs to CPS5 cluster (see Table 2), and D_{10} to CPS2 cluster, CPS2 and CPS5 are suggested to the designer as core *design fragments*. If the designer selects design fragment CPS2, the designer may select D_5 or D_{10} as the starting point. If the designer selects design fragment CPS5, the designer may select D_1 , D_2 , or D_7 as the starting point. The designer may extend his/her own designs from the instantiated design fragments suggested (CPS2 or CPS5), or from the complete designs suggested (D_5 , D_{10} , D_1 , D_2 , or D_7). Thus, for each case, several designs are immediately derived from a textual description.

For case 2, two concepts (C_2 and C_5) are identified from two keywords ‘company’ and ‘employee,’ and three designs (D_1 , D_2 , and D_4) are retrieved. For case 3, three important keywords exactly match C_5 . For case 2, three design fragments (CPS1, CPS4, and CPS5) are suggested and D_3 and D_9 (from CPS4), or D_7 and D_8 (from CPS5) are suggested as alternative designs. Since CPS1 has only one design (D_4 itself), no further alternative design is suggested. For case 3, two core design fragments, CPS4 and CPS5, are suggested and also D_3 and D_9 (from CPS4), or D_7 and D_8 (from CPS5) are suggested as alternative designs. More core design fragments and alternative designs may be suggested as the size of repository grows.

The results show that a repository of design fragments can be used to make immediate, reliable suggestions of relevant core design fragments and alternative designs for the development of a new design from a plain text problem description. Prior designs can be reused without much modification. Since variants of designs are stored in the repository, the designer can navigate the repository to find a design to fit to the new situation. When the size of repository is practically large enough, the number of designs in a specific domain can be expected to contain sufficient design variants so that the designer can find a suitable one.

6. Conclusion

We have proposed a new kind of reusable artifact, design fragments, and provided a methodology for building a repository of them for efficient reuse. The methodology focuses on retaining all prior designs, creating design fragments through clustering and storing them in an organized manner. The approach provides several advantages. First, no additional effort is required to build the reusable designs. When a designer generates a design, he/she does not have to be alerted for the future reuse of the design; for example, trying to find common rules to be reused for many slightly different situations. Second, since design fragments are indexed based on similarity measures, appropriate design(s) can be easily searched. Third, a design fragment is a whole design that is developed and indexed based on common patterns. It provides an immediate solution without much modification once an appropriate design is found that is appropriate for the new system. Fourth, a design fragment is a higher granularity than analysis patterns and has the same specificity as a domain model, which makes it easier for reuse. We are developing a prototype to implement the methodology, and plan to populate it with designs obtained from design sessions

using a web-based interface, which will be developed using Java™. Clustering algorithms are being developed using C++. The ontological classifications and instantiations corresponding to them are stored together with analysis patterns in the Pattern Library, which is implemented in a relational database. The prototype system incorporates an existing system [11] to implement the ontology for the new instantiations, which are classified into a proper ontology and, in turn, stored in the Pattern Library. Use of the prototype should lead to a repository that includes multiple indexed designs for many different application domains, and facilitate further testing.

References

1. Alexander, C., S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language*, Oxford University Press, New York, 1977.
2. Coad, P., D. North, and M. Mayfield, *Object Models: Strategies, Patterns, and Applications*, Prentice Hall, 1995.
3. Fowler, M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
4. Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
5. Michalski, R. S., "Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and Algorithm for Partitioning Data Into Conjunctive Concepts," *International Journal of Policy Analysis and Information Systems*, Vol. 4, 1980, pp. 219-243.
6. Michalski, R. S. and R. E. Stepp, "Learning from Observation: Conceptual Clustering," In *Machine Learning: An Artificial Intelligence Approach* by Michalski, R. S., J. G. Carbonell, and T. M. Mitchell (Eds.), Vol. 1, Morgan Kaufmann, Los Altos, CA, 1983, pp.331-363.
7. Mili, H. et al., "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, June 1995, pp. 528-562.
8. Purao, S. and V. Storey, "Intelligent Support for Selection and Retrieval of Patterns for Object-Oriented Design," In *Proceedings of the 16th International Conference on Conceptual Modeling (ER'97)*, Los Angeles, 3-6 November, 1997a.
9. Purao, S. and V. Storey, "APSARA: A Web-based Tool to Automate System Design via Intelligent Pattern Retrieval and Synthesis," In *Proceedings of the 7th Workshop on Information Technologies & Systems*, Atlanta, GA., Dec. 1997b, pp. 180-189.
10. Purao, S., V. Storey, and T. Han, "Improving Reuse-based System Design with Learning," *Working Paper*, 1998.
11. Storey, V., Dey, D., Ullrich, H., and Sundaresan, S., "An Ontology-Based Expert System for Database Design," *Data and Knowledge Engineering*, 1998.
12. Storey, V., H. Ullrich, and S. Sundaresan, "An Ontology to Support Automated Database Design," *Proceedings of the 16th International Conference on Conceptual Modeling (ER'97)*, Los Angeles, 3-6, November, 1997, pp.2-16.

An Algorithm to Extract Is_A Inheritance Hierarchies from a Relational Database

Nadira Lammari
CEDRIC laboratory
CNAM (Conservatoire National des Arts et Métiers)
292 Rue Saint Martin, 75141 PARIS Cedex 03, France
lammari@cnam.fr

Abstract. This paper presents an algorithmic approach for extracting Is_A inheritance hierarchies from a relational database. This approach can be integrated into a global process of reverse engineering of object-oriented or extended entity/relationship conceptual schemas. It uses the instances of the database. Its originality resides in its capability not only to derive concealed inheritances but also to retrieve existence constraints (constraints on null values) within and between relations.

1. Introduction

As opposed to the relational model, the extended Entity/Relationship model (EER) or Object-Oriented model (OO) deals with concepts allowing a natural and precise description of the real world. The generalization/specialization (or Is_A inheritance in object-oriented model) is one of these concepts.

Indeed, in the relational database model inheritances don't exist explicitly. They are expressed through different manners: 1) in the data, through the introduction of null values, 2) in the description of the database through inclusion constraints and views, 3) or also in the application programs especially through the description of cursors.

Their extraction from a relational database has three advantages. First, it allows us to discover hidden entities (entities that conceptually exist but are not explicitly translated in the database) getting closer to the reality. Secondly, the extraction can facilitate the migration to an object-oriented database. Finally, it makes more efficient and rich the integration of schemas in an heterogeneous context.

The purpose of this paper is to present an algorithmic approach to extract Is_A inheritance hierarchies. This approach takes place at the conceptualization phase of a relational database reverse engineering process. The conceptualization phase allows us to extract the conceptual schema of a relational database from: a schema described according to the Database Definition Language (DDL), queries or part of programs described according to the Data Manipulation Language (DML) and eventually from the database instances.

Our approach uses instances of the database relations. It can be applied either for each relation or for a set of relations. However, for this second case, to have significant results, we must supply the proposed algorithm with some other information such as the constraints between relations and the semantic links between attributes. These informations, if they are absent in the specification of the database, could be produced by one of the existing reverse engineering processes.

The remainder of this paper is organized as follows. The example we refer to

throughout the paper is presented in Section 2. We also give a practical overview of our approach in the same section. Section 3 describes the extraction of an Is_A inheritance hierarchy from a relation. Section 4 is the generalization of the process to a set of relations. In Section 5 we discuss the related works. Section 6 concludes the paper presenting issues for future research.

2. Example

This section presents an example allowing us to illustrate the algorithmic approach used in extracting Is_A inheritance hierarchies. It is related to a relational database for the management of an university. The schema associated to the database is as follows:
PERSON (AccessCode, LastName, FirstName, BirthDate, Address)
FACULTY (Faculty-Id, AccessCode, RecruitDate, Salary, Rank, Group, Step, Department, ResearchTeam)
STAFF (StaffCode, AccessCode, RecruitDate, Rank, Group, Step, Salary, Service)
STUDENT (Stud-Id, AccessCode, Level, AcademicCycle, ResearchTeam)
STUDENT-HIST (Stud-Id, Year, Result, AcademicCycle, Level)
RESEARCH-TEAM (ResearchTeam-Id, TeamName, LaboCode, TeamManager, Budget)
LABORATORY (LaboCode, Manager, YearBudget)
SERVICE (ServiceNumber, Manager, SupervisingService)
DEPARTMENT (DepartmentCode, Manager)
STRUCTURE (StructureCode, StructureName, Address)
S_S (StructureCode, ServiceNumber)
S_L (StructureCode, LaboCode)
S_D (StructureCode, DepartmentCode)

A data sample of the some tables is given in Annex 1 of this paper. The different integrity constraints (referential links and inclusion constraints) are described through the graphs depicted in Figures 1 and 2. Some of these constraints could be deduced, if they are not present in the database specifications, by the application of one of the existing reverse engineering processes.

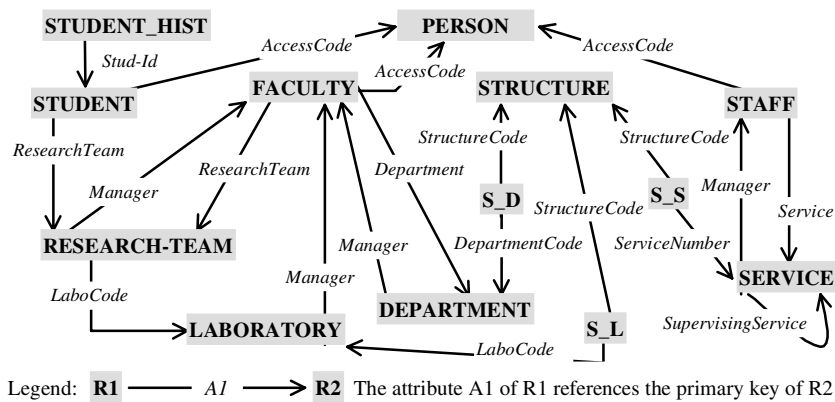


Figure 1 – Referential constraints between the relations of the university database

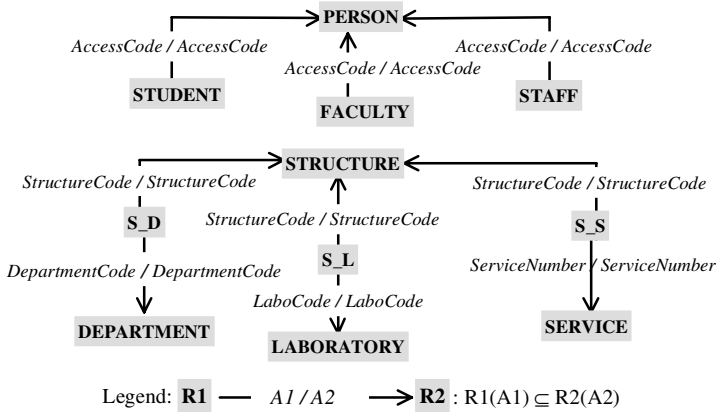


Figure 2 – Inclusion constraints between the relations of the university database

To show all the facets of our approach, we also suppose that the reverse engineering process to which our algorithm is integrated, supplies us with the following complementary information:

- STUDENT (AccessCode) $\cap$ FACULTY (AccessCode) $\neq \emptyset$.
- The attribute 'AccessCode' is a secondary key in STUDENT, FACULTY and STAFF.
- The attribute 'Group' takes four values (« APROF » for assistant professor, « ASSO » for associate professor, « FPRF » for full professor, « TASS » for temporary assistant) in the relation FACULTY. However, the reverse engineering process shows that the application is especially interested by assistant and full professors.
- The semantic links between attributes of relations related by inclusion or intersection constraints are described through Figures 3 and 4. For the readability of the graphs we only represent synonymy or identity links and homonymous links.

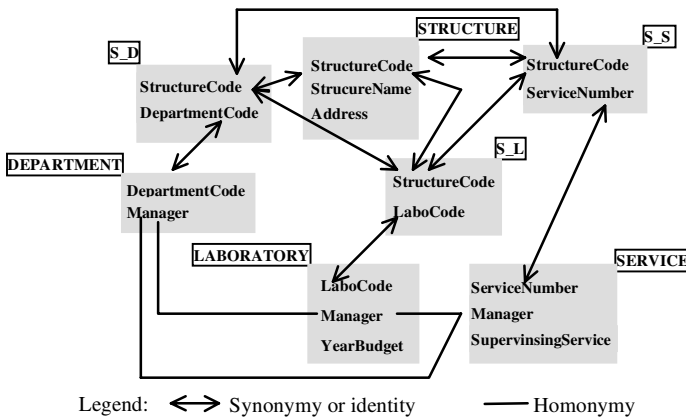


Figure 3 – Semantic links between the structural part of the university database

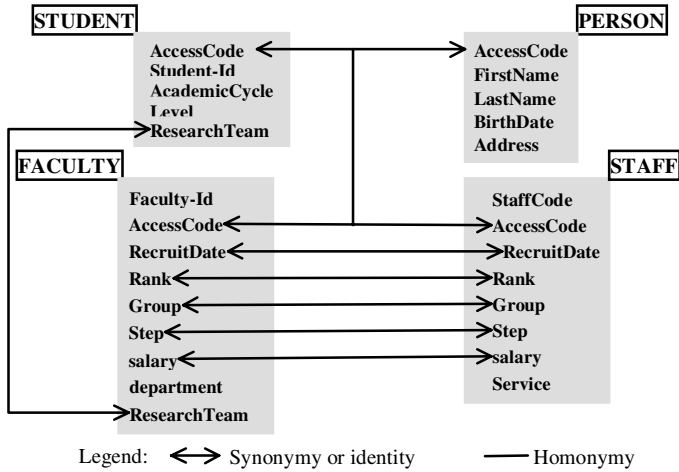


Figure 4 – Semantic links between the actors of the university database

The Is_A inheritance hierarchy reverse engineering algorithm is executed in two phases. First, according to the hypotheses, existence constraints are deduced from the database instances. Some of these constraints are described below:

- All the attributes of the relation *STUDENT*, except the attribute *ResearchTeam*, are mandatory.
- All the attributes of the relations *PERSON*, *STAFF*, *STRUCTURE*, *LABORATORY*, *DEPARTMENT*, *S_D*, *S_L* and *S_S* are mandatory.
- All the attributes of the relation *FACULTY*, except the attribute *Rank*, are mandatory.
- All the attributes of the relation *SERVICE*, except the attribute *SupervisingService*, are mandatory.
- The attributes *Faculty-Id* and *StaffCode* can not have a value for the same person.
- The attributes *Rank* and *ResearchTeam* can have a value for a same person only in the case where the attribute *Faculty-Id* has a value for this person.

These constraints are then used to derive the inheritance hierarchies. They are of two kinds. Constraints a, b, c, d are constraints within relations. The two last constraints are between relations. They can also be seen as constraints within views. For example, if a view *EMPLOYEE* gathering all the properties of an employee is defined in the database, then these two constraints are stated in this view.

In the second phase, the algorithm will derive the Is_A inheritance hierarchies presented in Figure 5. For that purpose it will use the splitting mechanism formally and completely described in [1] and for which we give a brief formal presentation in Section 3. An intuitive presentation is given below through its application to a simple example.

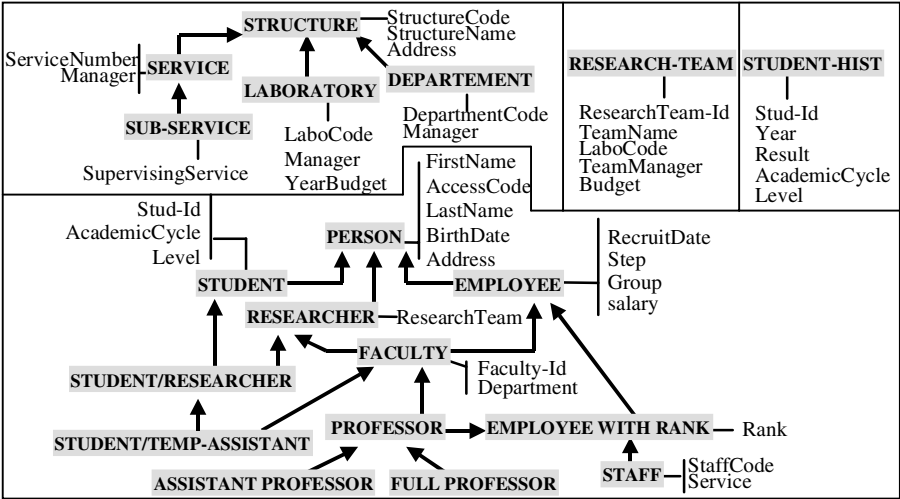


Figure 5 – The deduced Is_A inheritance hierarchies

2.1. Intuitive Description of the Splitting Mechanism

The idea carried by the splitting mechanism is first to find, among the relations, entities (subset of instances) that share the same structure and then to construct an inheritance hierarchy according to the inclusion links between the structures of the deduced entities. For instance, let *TouristicPlace* be a relation representing all the touristic places of a country such as museums, historical monuments, etc. Its structure is defined by the mandatory attributes *Name*, *Description*, *Address* and *EntranceFee* and by the optional attributes *Century* and *ClosingDate*. We suppose for this example that only museums have a closing date and that there is no museum that is also a historical monument. This hypothesis is translated into the relation *TouristicPlace* by adding the integrity constraint : « *Century* and *ClosingDate* can not have a value in the same tuple ». This constraint is called an exclusive existence constraints. As described above, the relation *TouristicPlace* gathers only three interesting subsets of instances. These subsets are not disjoint. The first one represents all the touristic places (it is the set of all instances of the relation). The second one represents all the museums and the third one all the historical monuments. Each of these subsets groups instances sharing the same subset of not null attributes. For example, all the museums have a name, a description, an address and a closing date. Therefore the relation *TouristicPlace* can be splitted into three entities *E1*, *E2*, *E3* associated to the three subsets of instances. This decomposition is based on the implicit existence constraints (those deduced from the type of the attributes (mandatory or optional)) and on the above explicit exclusive existence constraint. The resulting entities are organized into an inclusion graph (see Figure 6a), and by inverting the direction of the graph inclusion arrows, we obtain an *Is_A* inheritance hierarchy (see Figure 6b). The designer can thereafter give a name for each of these entities. For instance, he can call the entity *E1* ‘*TouristicPlace*’, the entity *E2* ‘*Museum*’ and the entity *E3* ‘*Historical Monument*’.

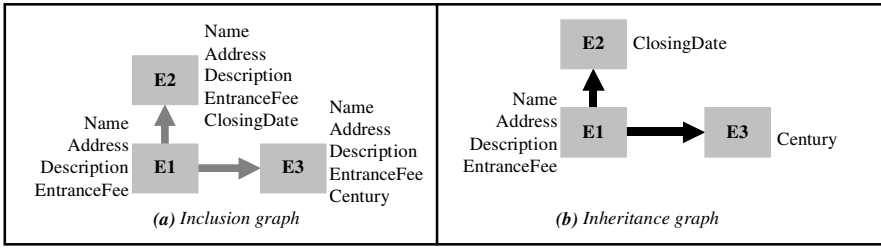


Figure 6 – The result of splitting the relation *TouristicPlace*

3. Extraction of an Is_A Inheritance Hierarchy from a Relation

A relational table usually corresponds to one or several conceptual entities. The cohabitation of these different entities in the same table is expressed by the introduction of null values and eventually by the definition of constraints on them. These constraints are also called existence constraints. They are of four kinds: non-null constraints, mutual, exclusive and conditional existence constraints.

A *non-null constraint* expresses the fact that a subset of the relation attributes set doesn't contain null values [2]. We say, in this case, that the attributes of this subset are mandatory attributes.

A *mutual existence constraint*¹ defined between two attributes x and y of a relation, denoted $x \leftrightarrow y$, describes the simultaneous presence and absence of x and y in the relation tuples. In other words, any tuple where x is null, y is also null and vice versa.

An *exclusive existence constraint*¹ defined between two attributes x and y of a relation, denoted $x \nleftrightarrow y$, translates the non coexistence of x and y in the relation tuples. In other words, any tuple where x is not null, y is null and vice versa.

Finally, a *conditional existence constraint*¹ defined between two attributes x and y of a relation, denoted $x \rightarrow y$, captures the fact that any tuple where x is not null, y is also not null. The opposite is not always true. We read this constraint « x requires y ». When x requires y and y requires x then x and y are related by a mutual existence constraint.

The identification of the different entities represented in a table is performed by analyzing the existence constraints. This technique is described in [1]. It is called a normalization process. It consists first in deducing, from the table structure (its attributes set), all the valid sub-structures (substructures that are compatible with the existence constraints) and then in organizing them into an inheritance hierarchy.

Let S be a subset of the attributes set of a relation R . S is said to be a valid sub-structure of R if and only if S satisfies the following three rules :

- *Homogeneity rule*: any attribute of R linked to an attribute of S by a mutual existence constraint is in S ,
- *Non-partition rule*: There is no exclusive existence constraints between attributes of S ,

¹ The formal definitions of these constraints and their generalization to a set of attributes can be found in [3] and [4].

- *Convexity rule*: any attribute of R required by a group of attributes of S is in S (taking into account the conditional existence constraints)

Since existence constraints are not always specified (as integrity constraints) in the database, we have to deduce them by analyzing the database instances. For this purpose, we proceed in two steps. First we determine a Boolean function describing all the different types of tuples encountered in the data. Then we deduce the existence constraints by the transformation of this Boolean function.

3.1. Determination of the Boolean Function

Let:

- $R(A_1, \dots, A_i, \dots, A_n)$ be the relation to be analyzed, E_R its extension (set of tuples), $T_{R,k}$ the k^{th} n -uplet of E_R and C its set of existence constraints.
- $S(x_1, \dots, x_i, \dots, x_n)$ be a relation which attributes are Boolean variables, E_S its extension and $T_{S,j}$ the j^{th} n -uplet of E_S . E_S is the cartesian product of the x_i domains (example : $E_{S(x_1, x_2)} = \{(0,0), (0,1), (1,0), (1,1)\}$).
- F be the application defined as follows :

$$F : E_R \rightarrow E_S$$

$$T_{R,k} \mapsto T_{S,j} \text{ such that } \forall A_i \text{ of } R, T_{R,k}(A_i) \neq \text{nul} \Leftrightarrow T_{S,j}(x_i) = 1$$

The image of E_R , denoted $F(E_R)$, is a subset of E_S . It is also the set of the different types of tuples that exist in the relation R (from a point of view of presence or absence of values for its attributes). In Figure 7 this set is represented by the dark spherical shape. Asserting that R contains at least one of the types of tuples of $F(E_R)$ means that the function representing these different tuples is true. For our example of Figure 7, $F(E_{\text{Service}})$ is equal to the set $\{(1, 1, 0), (1, 1, 1)\}$. This equality can be translated by the boolean function:

$$\phi(x_1, x_2, x_3) = x_1.x_2.\bar{x}_3 + x_1.x_2.x_3 = 1$$

Each minterm of ϕ describes a type of the tuples of R .

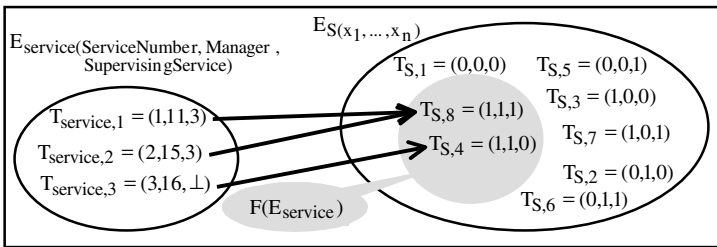


Figure 7 - Description of F for the relation *SERVICE*

3.2. Existence Constraints Derivation

$F(E_R)$ is the set of tuple types that exists in R . As a consequence, $F(E_R)$ satisfies the set C of existence constraints of R . In other words :

- if , for example, an attribute A_i is mandatory in R , then the associated variable x_i is equal to 1 for any tuple of $F(E_R)$.

So, $\varphi(x_1, \dots, x_n)$ can be written like this :

$$x_i, \varphi'(x_1, \dots, x_n) = 1,$$

- if two attributes A_i and A_j of R are linked by an exclusive existence constraint ($A_i \leftrightarrow A_j$), the project operation $\Pi_{x_i, x_j} F(ER)$ can not contain the couple (1, 1). Therefore, $\varphi(x_1, \dots, x_n)$ can be written:

$$(\bar{x}_i + \bar{x}_j) \varphi'(x_1, \dots, x_n) = 1,$$

- finally, if an attribute A_i requires an attribute A_j ($A_i \mapsto A_j$), then the project operation $\Pi_{x_i, x_j} F(ER)$ can not contain the couple (1, 0).

Therefore, $\varphi(x_1, \dots, x_n)$ can be written:

$$(\bar{x}_i + x_j) \varphi'(x_1, \dots, x_n).$$

For the generalization, we can say that if the attributes $A_i, \dots, A_j$ require the attribute A_k ($A_i, \dots, A_j \mapsto A_k$), then $\varphi(x_1, \dots, x_n)$ can be written:

$$(\bar{x}_i + \dots + \bar{x}_j + x_k) \varphi'(x_1, \dots, x_n) = 1.$$

For our example related to the relation SERVICE, $\varphi(x_1, x_2, x_3) = x_1.x_2.\bar{x}_3 + x_1.x_2.x_3 = 1$ can be written $\varphi(x_1, x_2, x_3) = x_1.x_2 = 1$. This last equality is verified when the two boolean variables are true (equal to 1) for any type of tuples of $F(E_{\text{Service}})$. This is only realized when the attributes ServiceNumber and Manager (attributes associated with these two variables) are not null in all the tuples of the relation SERVICE.

To summarize, we can assert that if we want to derive existence constraints from $\varphi(x_1, \dots, x_n)$, it is sufficient to convert this function into a conjunctive form and then to study each of its minterms. The transformation of φ from a disjunctive form to a conjunctive one can be done by applying the inversion rule of Shannon [5] which consists on calculating $\bar{\varphi}$. A minterm of $\bar{\varphi}$ of type « x_i » is translated to the non-null constraint « A_i is mandatory ». A minterm of type « $(\bar{x}_i + \dots + \bar{x}_j + x_k)$ » describes a conditional existence constraint « $A_i, \dots, A_j \mapsto A_k$ ». Finally, a minterm of type « $(\bar{x}_i + \bar{x}_j)$ » is translated to the exclusive existence constraint « $A_i \leftrightarrow A_j$ ». Two attributes A_i and A_j coexist in R ($A_i \leftrightarrow A_j$) if and only if the two conditional existence constraints $A_i \mapsto A_j$ and $A_j \mapsto A_i$ have been derived from $\bar{\varphi}$.

We therefore deduce the following algorithm for the extraction of an Is_A inheritance hierarchy from instances of a relation :

Begin

1. Derive from ER the existence constraints :

1.1 Calculate $F(ER)$

1.2 Deduce $\varphi(x_1, \dots, x_n)$ (φ is in a disjunctive form)

1.3 Transform φ into a conjunctive form

1.4 Search among the minterms of φ the existence constraints of R .

2. Derive from R , by analyzing existence constraints found in 1.4, the hierarchy H .

{use for this purpose the algorithm described in [1]}

End.

For our example using the relation SERVICE, the derived Is_A inheritance hierarchy (by application of the splitting algorithm and by taking into account the constraint «ServiceNumber and Manager are mandatory attributes »), is the following:



Figure 8 – Decomposition of the relation SERVICE

3.3. Remark

Even if a relation doesn't contain null values, it can be decomposed. Indeed, for the application requirements, we can specify views or programs to manipulate hidden entities. The latter are selected (in the views or programs) according to a value taken by an attribute of a relation. It is interesting in this case to represent these entities in the conceptual model. For example, suppose that, among the queries manipulating the relation FACULTY, exist some queries related to the selection of assistant and full professors. This selection is performed according to the values taken by the attribute Group. It would be desirable in this case to represent these entities (assistant and full professor) in the conceptual model.

In order to reuse the above technique of inheritance hierarchy extraction, we first modify the initial relation (FACULTY for our example) into a relation that can be therefore splitted according to null values. We apply the following transformation rule:

Let $R(..., A_i, ..., A_j ...)$ be a relation and $\{V_{i,1}, ..., V_{i,k_i}\}, ..., \{V_{j,1}, ..., V_{j,k_j}\}$ the subsets of values associated to $A_i, ..., A_j$. The transformation rule consists first in creating the relation $R'(..., A_i, \{V_{i,1}, ..., V_{i,k_i}\}, ..., A_j, \{V_{j,1}, ..., V_{j,k_j}\}, ...)$ and then in computing its extension according to the following algorithm:

```

Begin
For every tuple T of R Do
  Insert in R' the tuple :
     $T' = (... , T(A_i) , \underbrace{0, \dots, 0}_{k_i \text{ times}} , ..., T(A_j) , \underbrace{0, \dots, 0}_{k_j \text{ times}} , ...)$ 

  For every  $A_p \in \{A_i, \dots, A_j\}$  Do
    If  $T(A_p) = V \in \{V_{p,1}, \dots, V_{p,k_p}\}$  Then  $T'(V) = 1$  End_If
  End_For
End_For
End.
```

For our example the relation FACULTY is translated into the relation FACULTY' of Figure 9:

Faculty-Id	...	Group	APROF	FPROF	Echelon	...	ResearchTeam
2	...	TASS	⊥	⊥	1		2
4	...	APROF	APROF	⊥	3	...	1
8	...	APROF	APROF	⊥	2	...	1
13	...	TASS	⊥	⊥	1	...	3
14	...	APROF	APROF	⊥	1	...	3
19	...	FPRF	⊥	FPROF	1	...	2
27	...	FPRF	⊥	FPROF	3	...	4
28	...	TASS	⊥	⊥	1	...	3
30	...	APROF	APROF	⊥	2	...	4
32	...	ASSO	⊥	⊥	1	...	1
36	...	FPRF	⊥	FPROF	2	...	2

Figure 9 - Table *FACULTY'* issued from the transformation of the relation *FACULTY*

Finally, this algorithm amounts to create a BitMap² index for the values that may interest the application and then to merge it with the initial relation R. This algorithm can be applied to intervals of values. In this case, each $V_{i,j}$ is a range of values.

4. Generalization of the Extraction to a Set of Relations

Two relations $R_i(A_1,...,A_n)$ and $R_j(B_1,...,B_m)$ can be linked by one or several inclusion constraints of type $R_i[X] \subseteq R_j[Y]$ such that X (respectively Y) designates a sequence of k distinct elements of $\{ A_1,..., A_n \}$ (respectively of $\{ B_1,..., B_m \}$). If X (respectively Y) contains one of the candidate keys of R_i (respectively of R_j) then every entity represented by R_i is also an entity represented by R_j . Generally speaking, we can say : if $R_i[X] \cap R_j[Y] \neq \emptyset$ such that X (respectively Y) contains one of the candidate keys of R_i (respectively of R_j) then some entities represented by R_i are also entities represented by R_j .

Therefore, we can consider two different approaches to extract Is_A inheritance hierarchies from a set of relations. The first one consists in extracting from each relation its associated hierarchy and then on using a schema integration technique to deduce the final hierarchy. In this case the constraints are used to determine which hierarchies are to be merged. The second approach consists first in building the relation « Union » that regroups all the entities described by the related relations and then in the application of the above extraction technique. We have chosen in this paper to present the second approach because it derives not only the existence constraints within relations but also those between them. Some of the latter constraints are also constraints within views.

² A definition of a BitMap index can be found in [6]

4.1. Building of the Relation « Union »

The relation « Union » is built for each group of relations related by intersection or inclusion constraints (constraints defined using primary or candidate keys). These constraints are either specified in the database schema or deduced by any reverse engineering process. Furthermore, the building of the relation « Union » also requires the knowledge of the semantic links between the attributes of the group of relations. In other words, if two attributes are semantically equivalent (such as synonyms) only one will be in the description of the table « Union ». On the contrary, if they are semantically different (such as homonyms) they will both appear in the description of « Union ».

Let $R_1(A_{1,1}, \dots, A_{1,n_1}), \dots, R_i(A_{i,1}, \dots, A_{i,n_i}), \dots, R_m(A_{m,1}, \dots, A_{m,n_m})$ be relations related by intersection or inclusion constraints. Suppose that $A_{1,1}, \dots, A_{i,1}, \dots, A_{m,1}$ are their respective keys. The relation « Union » will have $\{B_1, \dots, B_m\}$ as its set of attributes. This set will regroup all distinct attributes (attributes that are semantically different) of the different relations. To compute its extension (its tuple set) we proceed as follows :

```

Begin
  For each tuple T of  $R_i$  Do
    Search in the table « Union » the tuple T' such that
       $T(A_{i,1}) = T'(B_j)$ ,  $B_j$  is synonym or identical to  $A_{i,1}$ 
    If T' is already in « Union »
      Then Modify T' with the values  $T(A_{i,2}), \dots, T(A_{i,n_i})$ 
    Else Insert, in the « Union », a new tuple T' by taking into
      account  $T(A_{i,1}), T(A_{i,2}), \dots, T(A_{i,n_i})$ .
      {T'(B_j) takes a null value if B_j doesn't have a
       corresponding attribute among those of  $R_i$ }
    End If
  End-For
End
  
```

In our example the relations SERVICE, STRUCTURE, DEPARTMENT, LABORATORY, S_S, S_D, S_L constitute a group related by inclusion constraints (see Figure 2). Their tuples (see Annex 1) are gathered in a unique relation « Union ». The specification of this relation takes into account the semantic links between the group attributes (see Figure 3). Figure 10 gives the extension of this relation.

Struct. Code	Struct. Name	Address	Labo Code	Labo. Manager	Year Budget	Service Number	Service Manager	Supervising Service	Depart. Code	Depart. Manager
1	Personnel	Paris	⊥	⊥	⊥	2	15	3	⊥	⊥
2	Accounts	Paris	⊥	⊥	⊥	1	11	3	⊥	⊥
3	Maths	Evry	1	36	70M	⊥	⊥	⊥	⊥	⊥
4	Maths	Paris	⊥	⊥	⊥	⊥	⊥	⊥	1	36
5	Info	Paris	⊥	⊥	⊥	⊥	⊥	⊥	2	4
6	Info	Bobigny	2	14	58M	⊥	⊥	⊥	⊥	⊥
7	Management	Paris	⊥	⊥	⊥	3	16	⊥	⊥	⊥

Figure 10 - Description of the relation UNION

5. Related Works

Since the eighties, we observe a growing interest in the reverse engineering of information systems ([7], [8], [9], [10], [11], [12], [13], [14]) especially for the re-engineering of relational databases to OO or EER conceptual schemas ([14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]). The main reasons of this increasing interest are for some people the growing maintenance costs, the missing (or no precise) documentation, the semantic degradations of their legacy systems and for the others the desire to migrate to OO databases with least costs.

Some of these reverse engineering approaches generate Is_A inheritance hierarchies. Most of them use heuristic techniques for the identification of inheritances. For instance, [26] considers three cases of Is_A inheritance identification:

- 1) the case of two relations having the same primary key and where the key of the first table refers to the one of the second table,
- 2) the case where null values occur in a table and where the null value of one of its attributes is conditioned by a value taken by its other attribute,
- 3) and finally the case of relations having the same primary key and on which a factorization process can be applied.

In [27], a generalization/specialization link is identified when two relations are linked by an inclusion constraint. [29] and [28] propose heuristics to identify inclusion constraints and then use these constraints to capture IS_A links.

To summarize, the proposed identification techniques allow us to exhibit inheritances between and within relations. Most of them derive one level of hierarchies. They essentially differ on the diversity of the sources they use (the database description including the specification of the views, its instances, existing application programs or frequent SQL queries) and on the abstraction levels they cover. From this point of view, the technique described in [28] is the most complete: it exploits all the sources and it covers all the levels starting from the de-optimization of the relational database.

Our technique differs from those proposed in the literature in the sense that it is an algorithmic technique and that it extracts not only *all concealed hierarchies levels* but also *existence constraints*. Discarding the fact that the proposed process is used to extract IS_A hierarchies, it can retrieve a part of missed semantics through the existence constraints.

6. Conclusion

This paper presents a method allowing us to detect generalization links from relational tables. This method can be used to retrieve inheritance hierarchies of OO or EER conceptual schemas. It can be integrated to any complete relational database reverse engineering approach. Its main feature is that it derives both inheritance hierarchies and existence constraints.

The quality of the results obtained by this method depends on the exhaustiveness of the set of data (enough data to represent all the real world situations) and on the chosen reverse engineering approach to which this method is integrated. This is the reason why we have chosen to integrate our algorithm into the MeRCI approach [28]. Indeed MeRCI exploits the data definition language specifications, the data manipulation language specifications and the database extension. It also covers all the

levels of a reverse engineering process starting from the de-optimization phase of the relational database [30]. Finally it allows us to gather all the necessary entries for our algorithm.

Our current work involves integrating the algorithm to the tool provided by MeRCI. We intend to extend it to the extraction of inheritances from navigational and multidimensional (datawarehouses) databases.

Annex 1

Table FACULTY

Faculty-Id	AccessCode	RecruitDate	Rank	Group	Step	Salary	Depart.	ResearchTeam
2	22	01-09-98	⊥	TASS	1	10K	2	2
4	15	01-09-95	1	APROF	3	12K	2	1
8	1	01-09-96	2	APROF	2	12K	2	1
13	9	01-09-98	⊥	TASS	1	10K	1	3
14	12	01-09-98	1	APROF	1	16K	1	3
19	20	01-09-94	1	FPROF	1	20K	2	2
27	26	01-09-95	2	FPROF	3	20K	1	4
28	6	01-09-98	⊥	TASS	1	10K	1	3
30	24	01-09-94	2	APROF	2	16K	1	4
32	30	01-09-98	1	ASSO	1	20K	2	1
36	4	01-09-93	1	FPROF	2	20K	2	2

Table LABORATORY

LaboCode	Manager	YearBudget
1	36	70M
2	14	58M

Table SERVICE

ServiceNumber	Manager	SupervisingService
1	11	3
2	15	3
3	16	⊥

Table STRUCTURE

StructureCode	StructureName	Address
1	Personnel	Paris
2	Accounts	Paris
3	Maths	Evry
4	Maths	Paris
5	Info	Paris
6	Info	Bobigny
7	Management	Paris

Table DEPARTMENT

DepartmentCode	Manager
1	36
2	4

Table S_L

Structure Code	Labo Code
3	1
6	2

Table S_S

Structure Code	Service Number
1	2
2	1
7	3

Table S_D

Structure Code	Department Code
4	1
5	2

Acknowledgements

The author would like to thank Jacky Akoka and Isabelle Comyn-Wattiau for their help and their fruitful discussions.

References

- [1] Lammari N., Laleau R., Jouve M., *Multiple viewpoints of Is_A Inheritance Hierarchies through Normalization and Denormalization Mechanisms*. Proceedings of OOIS'98 (International Conference on Object-Oriented Information systems), Springer-Verlag, Paris, September 9-11, 1998.
- [2] Atzeni P., Morfuni M., Functional Dependencies and Constraints on Null Values in Databases, Institute of System Information Analysis, Technical Report Technique N° R111, Italia, 1985.
- [3] Lammari N., Laleau R., Jouve M., *Deriving Normalized Is_A Hierarchies by Using Applicability Constraints*. CAISE'96. Lecture Notes in Computer Science n°1080, Springer-Verlag, Heraklion, Grece, May 20-24, 1996.
- [4] Lammari N., *Réorganisation des Hiérarchies d'Héritages dans un Schéma Conceptuel Objet*. Phd thesis, Conservatoire National des Arts et Métiers, October 24, 1996.
- [5] Schneeweiss W. G., *Boolean Functions with Engineering Applications and Computer Programs*. Springer-Verlag, Berlin Heidelberg, Germany, 1989.
- [6] Gardarin G., *Bases de Données : Objet et Relationnel*, Eyrolles 1999.
- [7] Casanova M, Amarel de Sa J, *Designing Entity Relationship Schemas for Conventional Information Systems*, in Proc. of Entity-Relationship Approach, 1983.
- [8] Davis K.H, Arora A.K, *Converting a Relational Database Model into an Entity-Relationship Model*, Proceedings 6th Int. Conf. on ER Approach, New York, USA, 1987.
- [9] Nilsson E.G, *The Translation of COBOL Data Structure to an Entity-Relationship Type Conceptual Schema*, Proceedings Entity-Relationship Approach, 1985.
- [10] Hainaut J.L, *Database Reverse Engineering : Models, techniques and strategies*, Proceedings Tenth International Conference on Entity-Relationship Approach, 1991.
- [11] Navathe S.B, Awong A, *Abstracting Relational and Hierarchical Data with a Semantic Data Model*, Proceedings of Entity-Relationship Approach: a Bridge to the User, Elsevier Science Publishers, 1988.
- [12] Winans J, Davis K.H, *Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model*, Proc. of Entity-Relationship Approach, 1990.
- [13] Hainaut J.L, Tonneau C, Joris M, Chandelon M, *Schema Transformation Techniques for Database Reverse Engineering*, Proceedings 12th Inter. Conf. on Entity-Relationship Approach, Arlington, Texas, 1993.
- [14] Batini C, Ceri S, Navathe S.B, *Conceptual Database Design : An Entity-Relationship Approach*, The Benjamin/Cummings Publishing Company, Inc., 1992.
- [15] Fonkam M.M, Gray W.K, *An Approach to Eliciting the Semantics of Relational Databases*, Proc. of 4th Int. Conf. on Advance Information Systems Engineering - CAISE'92, Springer-Verlag, 1992.
- [16] Markowitz V.M., Makowsky, J.A., *Identifying Extended Entity-Relationship Object Structures in Relational Schemas*, IEEE Transactions on Software Engineering, Vol 16(8), 1990.
- [17] Hainaut J.L, Cadelli M, Decuyper B, Marchand O, *Database CASE Tool Architecture : Principles for Flexible Design Strategies*, Proceedings 4th Int.

- Conf. on Advance Information Systems Engineering - CAiSE'92, Springer-Verlag, 1992.
- [18] Chiang R.H.L, Barron T.M, Storey V.C, *Performance Evaluation of Reverse Engineering Relational Databases into Extended Entity-Relationship Models*, in Proc. of the 12th Int. Conf. on ER Approach, Arlington, USA, 1993.
 - [19] Petit J.M, Kouloumdjian J, Boulicaut J.F, Toumani F, *Using Queries to Improve Database Reverse Engineering*, Proceedings 13th International Conference on ER Approach, Manchester, 1994.
 - [20] Andersson M., *Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering*, in Proceedings of the 13th Conf. on ER Approach, Manchester, UK, 1994.
 - [21] Signore O, Loffredo M, Gregori M, Cima M, *Reconstruction of ER Schema from Database Applications: a Cognitive Approach*, Proceedings 13th Int. Conf. on ER Approach, Manchester, UK, 1994.
 - [22] Premerlani W.J, Blaha M.R, *An Approach for Reverse Engineering of Relational Databases*, Communications of the ACM, Vol. 37 N°5, 1994.
 - [23] Jeusfeld M.A, Johnen U.A, *An Executable Meta-Model for Reengineering of Database Schemas*, Proceedings 13th Int. Conf. on ER Approach, Manchester, UK, 1994.
 - [24] Vermeer M.W.W, Apers P.M.G, *Reverse Engineering of Relational Database Applications*, Proceedings 14th Conf. on Object-Oriented and Entity-Relationship (OOER'95), Brisbane, Australia, 1995.
 - [25] Missaoui R., Gagnon J. M., Godin R., *Mapping an Extended Entity-Relationship Schema into a Schema of Complex Objects*. OOER'95, LNCS 1021, Gold Coast, Australia, December 95.
 - [26] Ramanathan S. et Hodges J., *Extraction of Object-Oriented Structures from Existing Relational Databases*. SIGMOD Record, vol 26, n°1, March 97.
 - [27] Fong J., *Converting Relational to Object-Oriented Databases*. SIGMOD Record, vol 26, n°1, March 97.
 - [28] Akoka J, Comyn-Wattiau I, *MeRCI : An Expert System for Software Reverse Engineering, Fourth World Congress on Expert Systems, Mexico, 1998*.
 - [29] Chiang R.H.L, Barron T.M, Storey V.C, *Reverse Engineering of Relational Database: Extraction of an EER model from a relational database*, Data and Knowledge Engineering Vol n°12, 1994.
 - [30] Comyn-Wattiau I, Akoka J, *Reverse Engineering of Relational Database Physical Schemas*, Proceedings 15th International Entity Relationship Conference, Cottbus, Germany, 1996.

DOTS: a Generic Infrastructure for Decision-Oriented Collaborative Task Support

Jacques Lonchamp

LORIA, BP 254, 54500 Vandœuvre-lès-Nancy, France
Jacques.Lonchamp@loria.fr

Abstract. This paper describes a generic infrastructure for supporting decision-oriented collaborative tasks, i.e., structured tasks in which argumentation and decision are important aspects (e.g., brainstorming, document outlining, review/inspection, conceptual map co-design, confrontation/merging of viewpoints). Basically, the system aims at supporting asynchronous work, but can also support “occasionally synchronous” work. The approach is mainly based on fine-grain modeling of decision-oriented collaborative tasks and the use of different assistance techniques: guidance (i.e., task performance assistance), argumentative reasoning, group awareness. The paper describes the project objectives, the conceptual meta model, the task modeling language, and the current java prototype. This work is a first step in a strategy for the “conceptual integration” of different kinds of cooperation support through an extended modeling framework that should merge coarse-grain modeling of workflow management systems and fine-grain modeling as described here.

1. Introduction

Cooperative work can take various forms: coordinated individual tasks, synchronous collaborative tasks, and asynchronous collaborative tasks. Currently, most computerized tools support a single form of cooperation. Workflow management systems coordinate individual tasks; they are generic infrastructures, parameterized by explicit process models. Synchronous groupware systems support collective work sessions; they are dedicated tools or toolkits for building them. Asynchronous groupware systems are also generally dedicated tools, or less frequently generic environments with explicit task models [6, 14, 24]. Integration of these different kinds of cooperation support is an important research topic and several solutions have been described. A first one considers the problem as a tool integration problem: integration of groupware systems within workflow management systems. This simple solution suffers obviously from the discontinuity of concepts and supports. A second approach solves a part of the problem by supporting both synchronous and asynchronous modes of work (e.g., [4, 20]). A third approach, more ambitious, is based on “conceptual integration” through an extended modeling framework: classical coarse-grain process modeling of workflow management systems is extended towards fine-grain modeling

of collaborative tasks (mainly asynchronous and structured ones). The work described in this paper follows this direction, without neglecting however the complementary idea of synchronous and asynchronous time modes integration.

Fine-grain modeling has been proposed in reaction against classical coarse-grain modeling, not necessarily in relation with cooperative work [15, 16, 19]. As stated in [15] a fine-grain process model considers actions or activities that relate to or manipulate elements of the product representation scheme. Thus guidance can take the form of a recommendation on what to do in order to advance the product construction. This is in contrast with models that are “unaware” of the representation schemes they manipulate and treat them as coarse-grain “vanilla” objects with no internal structure or semantics. This technique is often used in dedicated application domains like requirements engineering [17, 22], review/inspection [5, 14], design rationale [7, 12], collaborative learning [25]. Our approach is more generic and aims at covering a broad range of tasks that are recurrent in many workflows and cannot be modeled with the usual workflow modeling languages. This orientation is also related to the idea that asynchronous work can often replace with profit (cost/efficiency) synchronous work (it has been already demonstrated for review/inspections [11, 23]). However, it is important also to support “occasionally synchronous” work.

Among collaborative tasks, decision-oriented tasks have to be considered first, owing to their importance. All cooperative work generates conflicts that cannot be avoided, and which must be resolved by taking decisions after some discussions: for instance conflicts about different perceptions of the real world (during analysis tasks), conflicts about the priorities attached to different goals (during design tasks), conflicts about technical choices (during construction tasks). This idea is supported by theoretical analysis of group work. For instance, McGrath [13] distinguishes four cooperative activity types: “generate” (ideas or plans), “decide/choose”, “negotiate”, and “coordinate”. The first three are related to this decision-making orientation in a broad sense. From a more operational point of view, many collaborative processes already studied exhibit an important decision-making aspect: it is the case for co-authoring/brainstorming [21], review/inspection [5], and confrontation/merging of viewpoints [3]. These applications require argumentation and decision-making support and several synchronous or asynchronous dedicated groupware systems are available [2, 3, 5, 25]. It will be interesting to compare them with our more generic approach.

This paper has four sections. Section 2 summarizes the objectives of the project. Section 3 describes the conceptual meta model, where the abstractions underlying task models are defined. This meta model reflects the current implementation of the DOTS (“Decision-Oriented Task Support”) prototype. Section 4 presents the task modeling language. Section 5 describes briefly the architecture, functionalities and usage of the DOTS prototype. The conclusion outlines some promising future research directions.

2. The objectives

In this section, the main objectives and requirements are briefly summarized.

1. The system should support a small group of people (from two to less than ten) participating in a collaborative task, mainly distributed in time (asynchronous) and space; “occasionally synchronous” work should also be considered. The workflow aspect, i.e., the coordination with other individual or collaborative tasks is, for the moment, left outside the scope.
2. The system should support a range of tasks, through a generic infrastructure, parameterized by a task model; this model should be often the customization of a library model.
3. The system should provide an efficient assistance in three domains: guidance (i.e., task performance assistance), argumentation and decision assistance, and group awareness (both asynchronous and synchronous).
4. The project should provide a complete system for model development (editing, compiling, and verifying), system deployment support (installing and instantiating), and model execution.
5. In usual cases, it should be possible to generate a fully operational system from the task model and from the standard kernel. In more complex cases, the generated code should be only completed in some well-defined locations.
6. Both the entire infrastructure and the generated code should be Java code, mainly for taking advantage of its platform portability property.
7. The project should provide a library of generic task models for at least collective document outlining and conceptual map co-authoring, collective review/ inspection, collective confrontation/merging of viewpoints, and free argumentation (i.e., emulation of a synchronous/asynchronous argumentation groupware).

3. The conceptual meta model

In DOTS, a task model is described according to four perspectives: activity-oriented, decision-oriented, product-oriented, and organization-oriented. Of course these perspectives are not independent and there are many relationships between them.

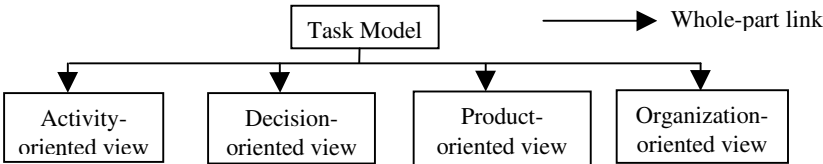


Fig. 1. The overall conceptual organization of a task model.

The next four subsections describe these perspectives and the last subsection emphasizes the argumentative reasoning aspect. We only discuss concepts that are implemented in the current prototype.

3.1. The activity-oriented view

A collaborative task is structured into *phases*, in which the nature of the work (see section 3.2), the number and the identity of the participants can vary. During a phase (individual or collective) decisions are taken that modify the *products* under construction: *operations* are the elementary chunks of work, triggered by a decision. During a phase, participants can also freely access *tools* for performing activities not related to decisions. The activity-oriented view of the task model mainly describes the phase types, the operation types, and the tool types.

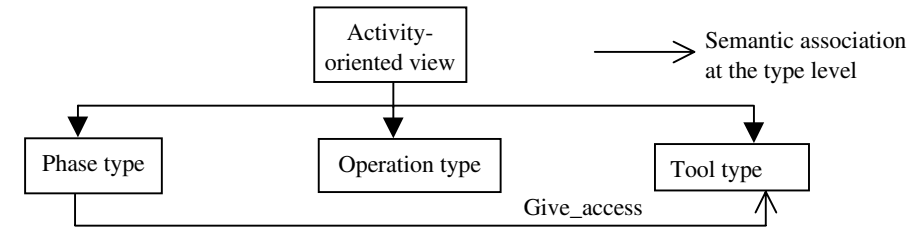


Fig. 2. Main elements of the activity-oriented view.

When the task model is instantiated, a graph of task instances is built, with *task precedence* links. This graph can include instances of the predefined ORPhase type, which models dynamical execution path choices.

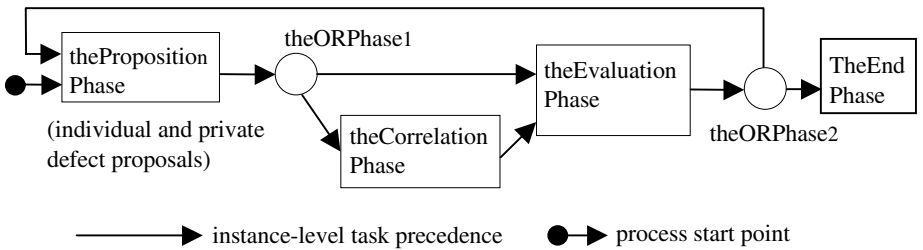


Fig. 3. A structured review/inspection task.

Figure 3 describes a structured review/inspection task. First, defects are proposed individually and privately. Then, a correlation phase can take place: it is decided within the first OR phase when they are too many similar proposals. In the evaluation phase, the (correlated) defects are collectively discussed and evaluated, i.e., accepted or rejected. At the end, an iteration can be decided within the second OR phase.

3.2. The decision-oriented view

Each phase is mainly defined by the subset of the task issue types taken in consideration at this stage. An *issue* is a problem that must be solved and which generally concerns the products under construction. In most tasks issue types are

progressively taken in consideration. Several *option* types specify how the issue type can be solved (e.g., AcceptDefect, and RejectDefect for EvaluateDefectIssue). At the level of the task execution, i.e., at the level of the instances, users argue about the options of each issue instance. The decision takes (more or less) in account this argumentation in relation with the resolution mode of the issue type (see below). *Arguments* are instances of a single Argument type and include a free textual rationale. Participants argue about the options and about the arguments themselves. They can also give qualitative *preference constraints* between the arguments (More ImportantThan or >, LessImportantThan or <, EquallyImportantThan or =). Participants can even argue about the constraints: constraints as arguments are refutable. The system always gives “the best solution” in accordance with the current argumentation state (see section 3.4), but the actual decision is kept independent from the argumentation.

To each option type can be associated an *operation* type. The operation is triggered when the option is chosen. This operation can modify:

- a product or one of its components (e.g. add a defect to the proposed defect list) ; this evolution can in turn raise new issues ;
- the task content (e.g. dynamical creation of issues, options, tools).

An option can also terminate a phase.

The main characteristic of an issue type is its resolution mode:

- individual: the issue is solved by its creator;
- individualPrivate: similar to the previous mode but the issue and its consequences are only visible by the creator of the issue;
- collectiveDemocratic: the resolution is collective because the solution is necessarily “the best solution” proposed by the system (see section 3.4) and at least two different users must take part in the argumentation;
- collectiveAutocraticWithoutJustification: the argumentation is collective but the choice is independent from the best solution proposed by the system and does not require any formal justification;
- collectiveAutocraticWithJustification: the choice is autocratic but requires a formal justification: an explanation step follows the argumentation and only the solver can argue during this step in order to make the best solution equal to his/her own solution.

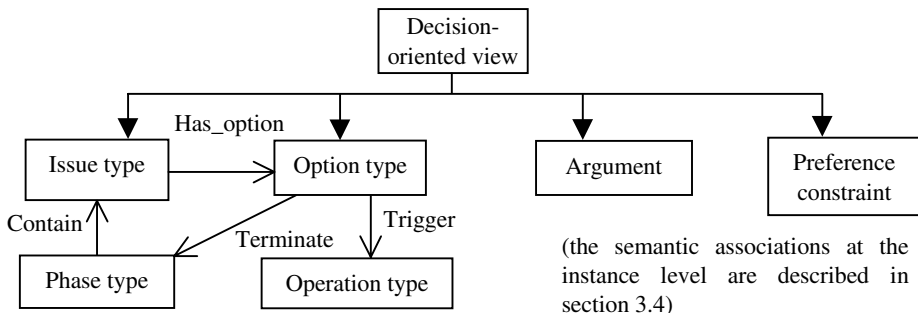


Fig. 4. Main elements of the decision-oriented view.

From a dynamical point of view, the life cycle of an issue is a sequence of interactions (expressed as regular expressions):

- RaiseIssue: create the issue instance (generally with parameters) and the corresponding option instances,
- (GiveArg | GiveConstr)+: creates the argumentation tree,
- SolveIssue: solves the issue and triggers the operation associated to the chosen option (this operation generally uses the issue parameters).

There exist two important simplified cases. For an individual issue with a single option, only RaiseIssue is necessary (all the remaining is automatic): the issue is just an elementary action. For an individual issue with several options, only RaiseIssue and GiveArg are necessary: the issue is just an individual choice between several elementary actions.

3. 3. The product-oriented view

A product includes *components* at different levels of granularity. Products are currently specialized into textual product, list product, graph product. A parallel classification exists for tools (e.g. textual viewers, list viewers, graph viewers). A minimum set of features is provided by these generic types (such as automatic graph layout methods; more specific features can be introduced through specialization, in the spirit of generic concept map editors such as [10]). Documents and tools can be instantiated either statically or dynamically.

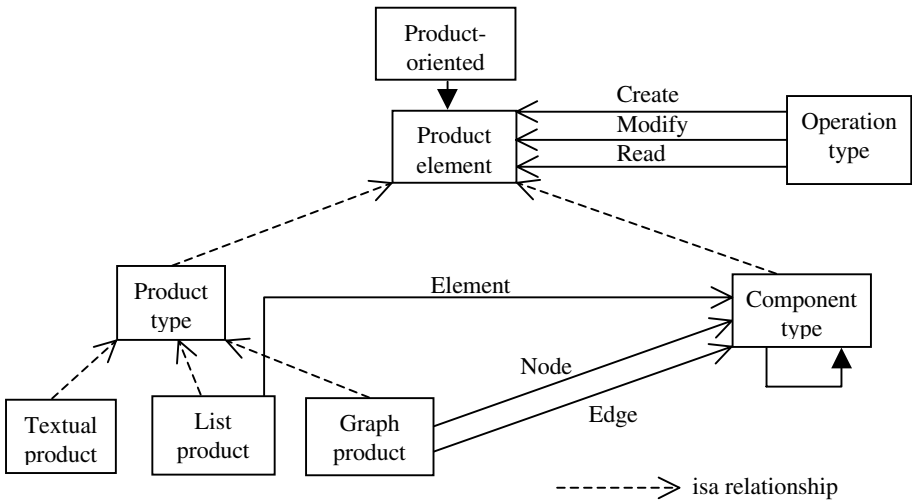


Fig. 5. Main elements of the product-oriented view.

3.4. The organization-oriented view

Actors (currently restricted to human participants) play *roles*. Role types define what actors are allowed to do. Actors are instantiated statically or dynamically.

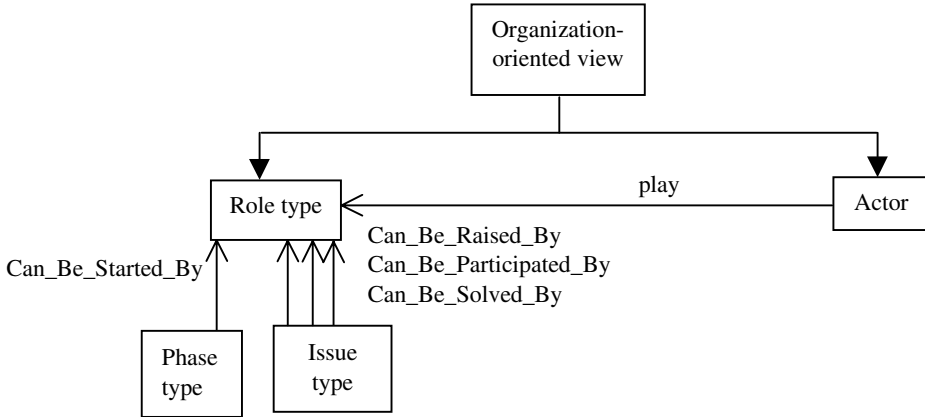


Fig. 6. Main elements of the organization-oriented view.

3.4. The argumentative reasoning aspect

The system provides participants means of expressing their individual arguments and qualitative preferences, the aim being the selection of a certain solution. We discuss the evaluation procedure in two steps related to the absence(presence) of qualitative preference constraints.

Without preference constraints

The issue, the options, the *arguments* “for” and “against” the options, the arguments “for” and “against” the arguments form an argumentation tree. A score and a status (active, inactive) that derive from the score characterize each node of this tree. The score of a father node is the sum of the weights of its active child nodes that are “for” him minus the sum of the weights of its active child nodes that are “against” him. If the score is positive the node is active otherwise it is inactive. Only status propagate in the tree (because scores have no global meaning). Without preference constraints all nodes have the same weight (for instance 5, middle of the arbitrary interval 0-10 used in the next subsection, where 10 denotes maximum importance). Leaves are always active. The preferred option (best solution of the issue - one or several) has the maximum score among all the options.

With preference constraints

Preference constraints are qualitative preferences between arguments of different options (global constraint) or between arguments of a same father argument (local constraint). One argument (source) is compared to the other (destination). To each constraint is associated a *ConstraintIssue* with 3 positions: *MoreImportantThan* ($>$), *LessImportantThan* ($<$), *EquallyImportantThan* ($=$). A constraint is active if both the arguments are active, if one of its options is chosen (score strictly higher than the others) and if it is consistent with the other constraints. Consistency is evaluated when the constraint is created and evaluated when another constraint becomes inactive (the evaluation is based on a path consistency algorithm). For instance, if arg_1 , arg_2 , and arg_3 have the same father, and $arg_1 > arg_2$, $arg_2 > arg_3$, then $arg_3 > arg_1$ is inconsistent. This last constraint becomes (provisionally) inactive.

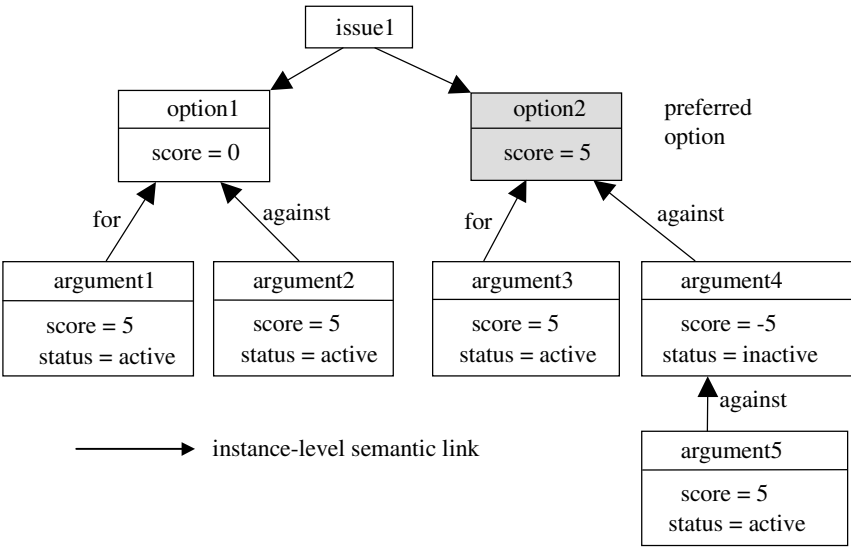


Fig. 7. An argumentation tree without constraint.

The weight of all arguments having the same issue as grand father (global constraint) or the same argument as father (local constraint) is computed by the following heuristics:

- all $>$ relationships are defined by propagating them along the $=$ relationships,
- for each argument arg involved in a constraint:
 - its max weight is computed, by subtracting 1 (starting from 10) for each arg_i such $arg_i > arg$;
 - its min weight is computed, by adding 1 (starting from 0) for each argument arg_j such $arg > arg_j$;
 - its final weight is computed as the average of its max and min weights.

- the weight of an argument not involved in any constraint is kept to the average value (5);
- the rules of section 3.4.1.for computing the scores and the status are applied with these computed weights.

After each modification the whole tree is re-evaluated: for instance, inactivating an argument can re-activate a constraint that was inactive because it was inconsistent with the former constraint, which changes the status of an argument, which propagates on the upper level, and so on. In figure 7 a constraint is added to the argumentation tree of figure 6. This argumentative reasoning technique is based on both Hermes and Zeno approaches [8, 9].

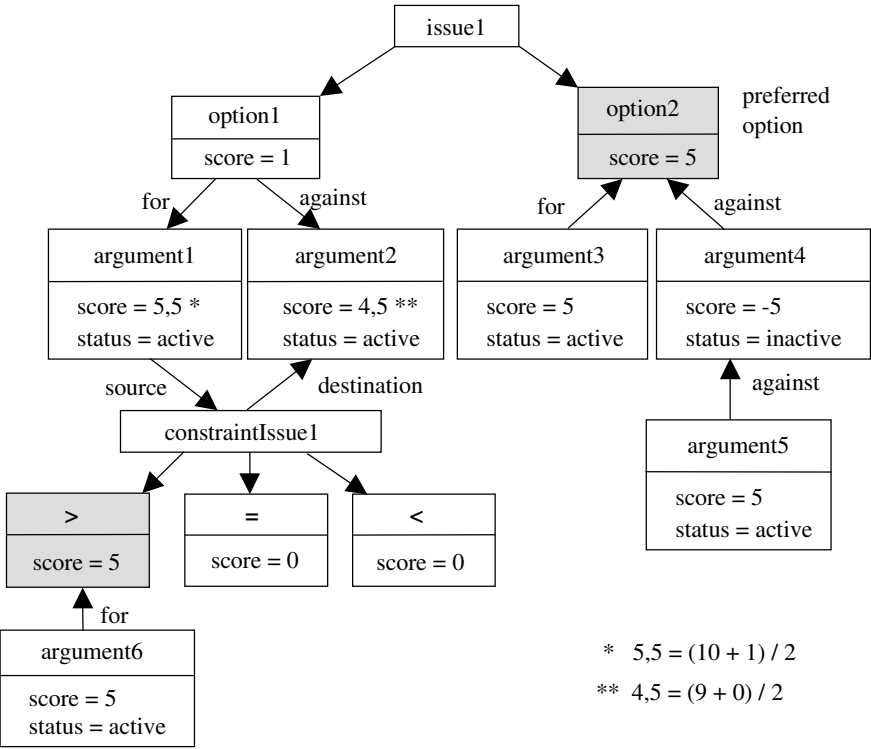


Fig. 8. The argumentation tree of Figure 7 after introducing a constraint.

4. The task modeling language

A task model is divided in two parts which describe the model specific entity types (specializations of phase, issue, option, role, document, tool, component, operation types) and the relationship types between them (Contain, Give_Access, Has_Option,

Trigger, Terminate, Can_Be_Started_By, Can_Be_Raised, Can_Be_Participated_By, Can_Be_Solved_By, Create, Modify, Read – see section 3).

The phase types, the option types, the role types are just characterized by their name. The issue types have in addition the resolution mode (see section 3.2), a boolean saying if the issue is a simple alternative (in this case all the argumentation takes place on a single option, otherwise it is necessary to argue “for” and “against” all the options), a boolean saying if only a single active instance can exist (e.g. it is the case for phase termination issues), and the parameters of the issue (interactions messages and possibly OQL queries for generating list boxes):

```
<issue-type-name>
  KIND <mode>
  TRUE_ALTERNATIVE <boolean>
  UNIQUE_ACTIVE_INSTANCE <boolean>
  PARAMETERS
    (LABEL <message> [QUERY <query>] ";" ) *
  END_PARAMETERS
```

The document types and the tool types are classified by content: text, list (giving the component type name), graph (giving the node type name, and the edge type name). The component types can have attributes:

```
ATTRIBUTE_TYPES
  (<type-name> <attribute-name> ["=" <constant>] ";" ) *
END_ATTRIBUTE_TYPES
```

The operation types are described through a list of elementary action specifications (of type CREATE, MODIFY, or DELETE – see the examples below):

```
ACTION
  (<action-specification> ";" ) *
END_ACTION
```

For instance, the EvaluateDefectIssue type is defined by:

```
EvaluateDefectIssue
  KIND collectiveDemocratic
  TRUE_ALTERNATIVE true
  UNIQUE_ACTIVE_INSTANCE false
  PARAMETERS
    LABEL "Give the defect name: " // parameter 1
    QUERY SELECT * FROM Model.DefectExtent WHERE
      status = "proposed";
  END_PARAMETERS
```

Two options types are associated to EvaluateDefectIssue: KeepDefect, and RejectDefect. RejectDefect can trigger an operation of type InvalidateDefect:

```
InvalidateDefect
  ACTION
    MODIFY Defect PARAM(1) WITH status = "refused";
    // PARAM(1) is a reference to the parameter 1
    // of the associated issue type
  END_ACTION
```

This is a very simple case. Several actions can also be related through local variables. The second example below shows the dynamical creation of a component (a cluster of ideas during a brainstorming), of a graph document associated to this component, and of a viewer tool associated to this graph:

```
AddCluster
ACTION
    CREATE Cluster AS cl WITH IName = PARAM(2)
                                // local variable cl
    WITH enclosingCluster = PARAM(1);
    CREATE ClusterGraph AS cg WITH father = cl
                                // local variable cg
    WITH fatherAttribute = "enclosingCluster"
    WITH componentIconPath = "Images/Cluster.GIF";
    CREATE ClusterGraphViewer WITH TheGraph = cg
    WITH IName = PARAM(3);
END_ACTION
```

The local variables `cl` and `cg` are useful for linking the three dynamically created entities.

In the current prototype, action specifications are not a complete data manipulation language. When the expressive power is insufficient, the tool designer must complete the target Java code produced by the DOTS compiler. Another possibility, more generic, is to specialize document types and tool types for defining or refining the semantics of frequently used actions.

5. The DOTS prototype

5.1. The system architecture

The system has a client/server architecture around an object database with a Java API (POET). The database provides persistency, consistency, safety, and security. Communication and notification aspects are managed by a specific Java infrastructure (so the demo version of the ODBMS is sufficient). Persistency is conforming to the ODMG Java binding: persistent classes are declared statically and pre-processed before the Java compiler is called. This makes impossible dynamical schema evolutions. So, in the current prototype, we have chosen to generate (transparently) one separate database for each task model and for each version of a task model. All these databases (generally quite small) are accessed through a «superbase», and can be located on different machines. Each database contains one task model version and all the task instances conforming to this model. If a task model is changed, it is possible to run instances of the two different versions located in the two different databases.

The client is independent of the task model. It is written in Java and swing. The development and deployment environment includes three other tools, all written in

Java and swing: an editor/compiler, an instantiator, and a statical verifier for instantiated models.

5.2. The database organization

Each database includes two parallel hierarchies of classes and «metaclasses». The application-specific classes and meta classes are specializations of the kernel classes. Kernel classes implement the basic mechanisms of the infrastructure: execution, guidance, group awareness, argumentative reasoning, and so on. The task model is expressed at the level of the instances of the metaclasses. These instances are persistent. For example, `MProposalPhase` is a specialization of the kernel metaclass `KMPhaseClass`, and `MProposeDefectIssue` is a specialization of the kernel metaclass `KMIssueClass`; the Contain relationship of the task model (see Figure 4), saying that `ProposeDefectIssue` is an issue type of `ProposalPhase` type is implemented between an instance of `MProposalPhase` and an instance of `MProposeDefectIssue`.

The task elements and the resulting products and components are instances of the classes and are obviously also persistent objects. For instance, `ProposalPhase1` instance and `EvaluationPhase1` instance are two specializations of the kernel `KPhaseClass`. The precedence relation between these two phases is expressed between these two instances.

There exists a bi-directional link between each instance of a metaclass and the corresponding class in the other hierarchy.

The DOTS compiler generates the task specific classes and metaclasses, the configuration file for ensuring persistency and the scripts for creating and installing the database.

5.3. The main functionalities

The user enters the system with a registered user name (created with the instantiator and kept in the «superbase »), in one of the task instances in which he/she plays a role. The user can then act in accordance with the task model, the current task status, and his/her role: he/she can start a phase, raise an issue, give an argument or a constraint, try to solve an issue, and so on. The user receives a threefold assistance: guidance (how to perform the task), argumentation and decision assistance, synchronous and asynchronous group awareness.

For guidance, the user can obtain the list of possible next interactions in accordance with the current task status and his/her role. Obviously only those possible interactions are accepted by the client. The user can also access to different textual and graphical views of the task model and of the task history (with colors highlighting for instance the active elements).

At the argumentation level, the best option of each open issue is shown in color in the graphical representation and also the active arguments and constraints; scores and weights can be displayed. The user can also list all open issues that are currently inconclusive (no option with a higher score than the others).

The main mechanism for asynchronous awareness shows what has evolved since the last connection of the same user in the same task (textual list, and specific color in all graphical views). For “occasionally synchronous” work, the user can obtain the list of all the connected users in the same task, can receive the notification of all constructive public actions from these other users in a notification window, and is warned when a document or a graphical representation becomes out of date (its background color changes).

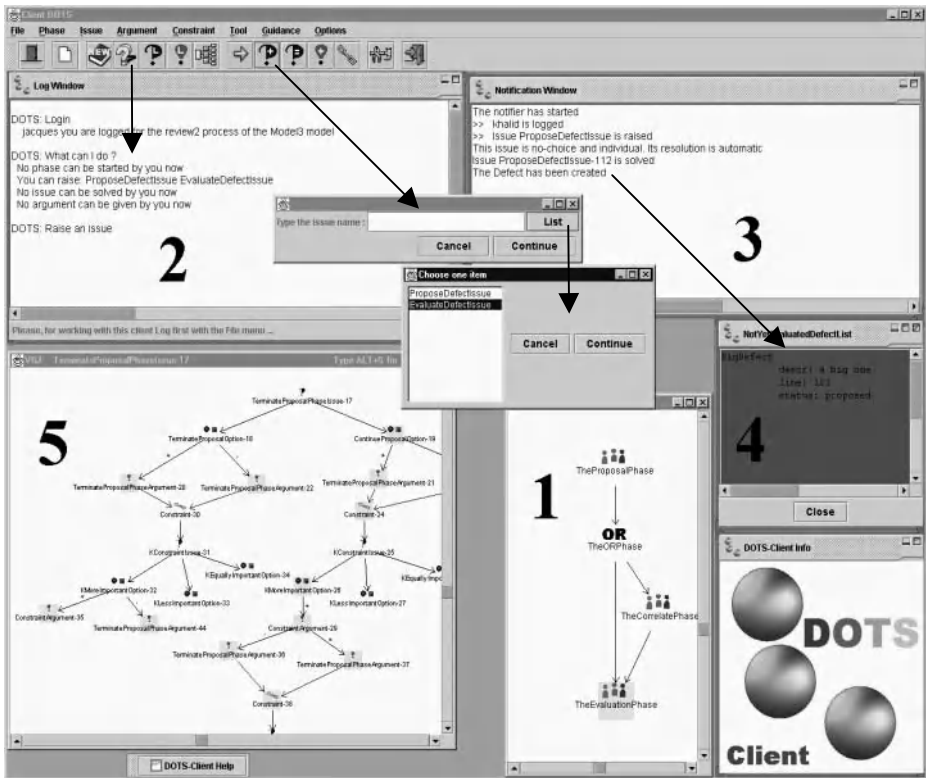


Fig. 9. The DOTS client.

Figure 9 shows a client during the evaluation phase of a simple review, whose model is shown in window 1. Window 2 is the log window that contains the results of the interactions (here, a «what can I do?» request). Window 3 is the notification window: one can see that another user has logged in and has proposed a new defect. Window 4 is the NotYetEvaluatedDefect viewer tool and its content has become out of date after the creation of the new defect (its background color is dark). Window 5 shows graphically the current state of an issue; this description as the task description in window 1 is up to date (white background). Icons with a colored square highlight active phases in window 1 and active nodes in window 5.

6. Conclusion

The generic infrastructure described in this paper aims at assisting participants of decision-oriented collaborative tasks. The approach is based on fine-grain modeling of these tasks and the use of different assistance techniques: guidance, argumentative reasoning, group awareness. The system makes a synthesis of classical features of generic process-centred environments on the one hand and of synchronous and asynchronous groupware systems on the other hand. A previous mock-up system written in Smalltalk had already convinced us of the approach interest, in particular through a real size experiment [11]. The fundamental “issue/argument/decision/operation” cycle seems easy to understand and use for inexperienced end users.

There are many improvement perspectives. In the short term, the library of generic models must be enriched: currently, it contains several basic brainstorming and review/inspection models, and a free argumentation groupware tool emulation. A more complex document outlining model is under development for demonstrating the reflective capabilities of DOTS: within “refinement tasks”, operations allow to develop the graph of task instances dynamically and under the control of some participants. We also plan to develop a model for viewpoints merging, in the spirit of [3]. The library of generic document and tool types has to be improved in parallel, in particular with hypermedia documents and more elaborated generic conceptual maps [10]. With this last extension DOTS could be specialized into a process-centred support system for collaborative conceptual modeling, and could be used for studying different models. Finally, the prototype itself could be improved in several ways: for instance, by developing a generic persistency service for making DOTS independent from the underlying persistence technology (Java serialization, RDBMS, ODBMS, orthogonal persistency).

In the long term, the central idea is to make concrete the “conceptual integration” approach that should merge coarse-grain modeling and fine-grain modeling. A parallel project in our research team aims at building a transactional workflow system [1] on top of a distributed configuration management system, written in Java. This kind of solution increases the importance of collaborative tasks such as review/inspection and confrontation/merging. We plan to carry out our “conceptual integration” work in this context.

References

1. G. Canals, C. Godart et al., A criteria to enforce the correctness of individually cooperating applications, *Information Sciences* (110) 3-4, Elsevier, 1998, pp 279-302.
2. J. Conklin, M. Begeman, M.L., gIBIS: A Hypertext Tool for Exploratory policy Discussion, *ACM Trans. on Office Inf. Systems*, 4, pp. 303-331, 1988.
3. S. Easterbrook, Handling Conflict Between Domains Descriptions through Computer-Supported Negotiation, *Knowledge Acquisition: an Int. Journal*, 3, pp 255-289, 1991.
4. A. Haake, B. Wilson, Supporting collaborative writing of hyperdocuments in SEPIA, in *Proc. CSCW'92*, Toronto, pp 138-146, 1992.

5. P. Johnson, D. Tjahjono, Improving Software Quality through Computer Supported Collaborative Review, in Proc. of third European Conf. on CSCW, Milan, 1993.
6. S. Kaplan, W. Tolone, D. Bogia, C. Bignoli, Flexible Active Support for Collaborative Work with Conversation Builder, in Proc. CSCW'92, Toronto, pp 378-385, 1992.
7. M. Klein, Capturing Design Rationale in Concurrent Engineering Teams, IEEE Computer, pp 39-47, January 1993.
8. N. Karacapilidis, D. Papadias, A Group Decision and Negotiation Support System for Argumentation Based Reasoning, in Learning and reasoning with complex representations, LNAI, Springer-Verlag, 1997.
9. N. Karacapilidis, D. Papadias, T. Gordon, An Argumentation Based Framework for Defeasible and Qualitative Reasoning, in Advances in Artificial Intelligence, LNAI 1159, Springer verlag, pp 1-10, 1996.
10. R. Kremer, Constraint Graphs: A Concept Map Meta Language, PhD Thesis, Univ. Of Calgary, 1997.
11. J. Lonchamp, B. Denis, Fine-Grained Process Modelling for Collaborative Work Support: Experiences with CPCE, 7th Mini EURO Conf. on DSS, Groupware, Multimedia and EC, Bruges, 1997 (to appear in Journal of Decision Systems, Hermès)..
12. J. Lee, Extending the Potts and Burns Model for Recording Design Rationale, in Proc. 13th ICSE, Austin, pp 114-125, 1991.
13. J.E. Mc Grath, (1984) Groups: interaction and performance, Prentice Hall, 1984.
14. F. Macdonald, J. Miller, Automatic Generic Support for Software Inspection, Technical Report RR-96-198, University of Strathclyde, Glasgow, 1996.
15. B. Nuseibeh, A. Finkelstein, J. Kramer, Fine-Grain Process Modelling, In Proc . 7th IWSSD, Redondo Beach, IEEE Press, 1993.
16. C. Potts, A Generic Model for Representing Design Methods, in Proc. 11th ICSE, Pittsburgh, pp 199-210, 1989.
17. C. Potts, K. Takahashi, A. Anton, Inquiry-Based Requirements Analysis, IEEE Software, pp 21-32, March 1994
18. G. Rein, C. Ellis, rIBIS: a real-time group hypertext system, Int. J. Man-Machine Studies, 34, pp 349-367, 1991.
19. C. Rolland, L'ingénierie des processus de développement de systèmes : un cadre de référence, Ingénierie des Systèmes d'Information, 4, 6, 1996.
20. M. Sohlenkamp, G. Chwelos, Integrating Communication, Cooperation, and Awareness: The DIVA Virtual Office Environment, In Proc CSCW'94, Chapel Hill, pp 331-343, 1994.
21. M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning, L. Suchman, Beyond the Chalkboard, CACM, 30, pp 32-47, 1987.
22. S. Si-Said, C. Rolland, G. Grosz, MENTOR: A Computer Aided Requirements Engineering Environment, in Proc. CAISE'96, LNCS 1080, pp 22-43, 1996.
23. D. Tjahjono, Comparing the cost effectiveness of GroupSynchronous Review Method and Individual Asynchronous Review Method using CSRS: Result of Pilot Study, Tech. Report, ICS-TR-95-07, University of Hawaii, 1995.
24. D. Tjahjono, Building Software Review Systems using CSRS, Tech. Rep. ICS-TR-95-06, Univ. of Hawai, Honolulu, 1995.
25. D. Wan, P. Johnson, Computer Supported Collaborative Learning Using CLARE: the Approach and Experimental Findings, in Proc. CSCW'94, Chapell Hill, pp 187-198, 1994.

Dockets: A Model for Adding Value to Content

Joachim W. Schmidt and Hans-Werner Sehring

Software Systems Institute (AB 4.02)
Technical University Hamburg-Harburg
<http://www.sts.tu-harburg.de>

Abstract. Dockets¹ are traditional paper means for supporting specific classes of workflows which concentrate on content and its value for individuals and organizations. A docket prescribes and finally documents some content reviewing process which is part of a larger value chain. Since contents represent essential investments they are carefully protected by principles like autonomy and ownership. However, value chains over contents materialize only in open and cooperative scenarios. Therefore, dockets aim at mediating carefully between open content availability for value-adding processes and control for investment protection. In this paper we present *dockets* as an innovative *conceptual model* for investment-protecting value chains over content. The model is introduced essentially by graphical means and is substantiated by our first experience with docket-driven application development. We demonstrate how our basic docket model scales to support general content management.

1 Introduction

Information is an increasingly valuable resource for both information providers and consumers. In the rapidly extending market for networked information services, information resources represent the essential value and investment of many enterprises. Owners of information resources, individuals as well as organizations, become more and more conscious to whom and under which conditions they provide access to their content. Resource autonomy is a highly ranked quality.

On the other hand, most information resources become and remain valuable only if they are subject to regular use and to value-adding processes. However, since such efforts may severely interfere with resource autonomy and ownership, they have to be realized by carefully monitored and agreed-upon processes. Furthermore, adding value to content nearly always requires some deeper insight into the larger value chains in which the resource and its content is involved.

¹ Docket

- a piece of paper accompanying or referring to a package or other delivery, stating contents, delivery instructions etc.;
- a summary of contents, as in a document;
- a list of things to be done;
- sometimes serving as a receipt.

[Collins Dictionary of the English Language, Collins London & Glasgow, 1981]

As a consequence, there is an increasing demand for support of a specific class of workflows which implement investment-protecting value chains over content. Such content-oriented value chains have to mediate carefully between possibly conflicting interests of the contributing domains:

- reliable content protection as demanded by resource content providers,
- liberal resource access as required by content reviewing experts, and
- goal-oriented exploitation of reviewing results by value chains over content.

In this paper we concentrate on the conceptual modeling [2] of those workflows intended to add value to content. For this purpose we analyze, abstract and apply *dockets*, which is a proven concept from traditional content management. Offices and organizations use paper docket if they want to initiate and finally document some routine, *re-view*-like operation on a piece of content which is attached to or referred by the docket. Furthermore, each docket addresses some review performer by whom it is returned to sender upon review completion.

In section 2 we motivate the concept of docket and sketch a model by giving some of the major abstractions provided by docket.

Section 3 discusses docket dynamics in terms of performer binding, property handling and docket tracing. In section 4 we apply our basic docket model to an extended range of applications.

The docket life cycle is discussed in section 5. First experience with docket-driven application development is reported in section 6 together with the emerging architecture of a generic framework for docket management. In section 7 we refer to related work before we make some concluding remarks in section 8.

2 Dockets as a Conceptual Model

Traditionally, docket are used for those (parts of) value chains in content management which follow well understood and somehow repetitive content-oriented interaction patterns. Typical docket examples are version update and management of reports, classification and annotation tasks on library material, or management and reviewing of conference submissions.

2.1 Dockets: A Proven Concept

In many cases content providers give access to their material only if they understand and agree why, how and by whom the content is used. On the other hand, reviewers working on contents and possibly contributing to their value expect a clear insight into the larger value chain within which their result will be evaluated and exploited. It is essential for content management to clarify the assumptions which performers participating in a docket scenario make about their roles and mutual relationships. Docket design makes such assumptions and roles explicit, and docket enactment supports their achievement.

Fig. 1(a) shows an example of a paper docket *template* for the periodic reviewing of our department's Research Activity Report. Docket templates have to meet *by design* the interests of all three domains and their performers:

Dean's Office
 Department of Computer Science, TUHH
 Phone:
 Fax:
 e-mail:

Subject: **Production/Publication of our Research Activity Report**
 To: Institute
 From: Mr./Ms./Dr. (Report Reviewer)
 Date: (Review Evaluator)

Dear CS colleagues;
 Please acknowledge the copy of your institute's latest contribution to the

Computer Science Research Activity Report

Last update of your contribution:
 Update information provided by:
 Published by TUHH until version:

Please review content and provide update information (if available) on

☐ External Projects	Update information available? ☐ yes ☐ no
☐ Industrial Contacts	Update information available? ☐ yes ☐ no
☐ Dissemination	Update information available? ☐ yes ☐ no
☐ Update Information:	 (number)

..... (attach pages if necessary)

..... (Initials)

Please return ☐ as soon as possible ☐ not later than

Please acknowledge receipt:

Yours sincerely,
 (Dean's Office)

(a) Paper Docket Template for Content Review

Top Cover *Page 24-25*
Bottom Cover *Page 26-27*
 Inside Front Cover *Page 28-29*
 Inside Back Cover *Page 30-31*

Table of Contents *Page 32-33*
Introduction *Page 34-35*
Chapter 1 *Page 36-37*
Chapter 2 *Page 38-39*
Chapter 3 *Page 40-41*
Chapter 4 *Page 42-43*
Chapter 5 *Page 44-45*
Chapter 6 *Page 46-47*
Chapter 7 *Page 48-49*
Chapter 8 *Page 50-51*
Chapter 9 *Page 52-53*
Chapter 10 *Page 54-55*
Chapter 11 *Page 56-57*
Chapter 12 *Page 58-59*
Chapter 13 *Page 60-61*
Chapter 14 *Page 62-63*
Chapter 15 *Page 64-65*
Chapter 16 *Page 66-67*
Chapter 17 *Page 68-69*
Chapter 18 *Page 70-71*
Chapter 19 *Page 72-73*
Chapter 20 *Page 74-75*
Chapter 21 *Page 76-77*
Chapter 22 *Page 78-79*
Chapter 23 *Page 80-81*
Chapter 24 *Page 82-83*
Chapter 25 *Page 84-85*
Chapter 26 *Page 86-87*
Chapter 27 *Page 88-89*
Chapter 28 *Page 90-91*
Chapter 29 *Page 92-93*
Chapter 30 *Page 94-95*
Chapter 31 *Page 96-97*
Chapter 32 *Page 98-99*
Chapter 33 *Page 100-101*
Chapter 34 *Page 102-103*
Chapter 35 *Page 104-105*
Chapter 36 *Page 106-107*
Chapter 37 *Page 108-109*
Chapter 38 *Page 110-111*
Chapter 39 *Page 112-113*
Chapter 40 *Page 114-115*
Chapter 41 *Page 116-117*
Chapter 42 *Page 118-119*
Chapter 43 *Page 120-121*
Chapter 44 *Page 122-123*
Chapter 45 *Page 124-125*
Chapter 46 *Page 126-127*
Chapter 47 *Page 128-129*
Chapter 48 *Page 130-131*
Chapter 49 *Page 132-133*
Chapter 50 *Page 134-135*
Chapter 51 *Page 136-137*
Chapter 52 *Page 138-139*
Chapter 53 *Page 140-141*
Chapter 54 *Page 142-143*
Chapter 55 *Page 144-145*
Chapter 56 *Page 146-147*
Chapter 57 *Page 148-149*
Chapter 58 *Page 150-151*
Chapter 59 *Page 152-153*
Chapter 60 *Page 154-155*
Chapter 61 *Page 156-157*
Chapter 62 *Page 158-159*
Chapter 63 *Page 160-161*
Chapter 64 *Page 162-163*
Chapter 65 *Page 164-165*
Chapter 66 *Page 166-167*
Chapter 67 *Page 168-169*
Chapter 68 *Page 170-171*
Chapter 69 *Page 172-173*
Chapter 70 *Page 174-175*
Chapter 71 *Page 176-177*
Chapter 72 *Page 178-179*
Chapter 73 *Page 180-181*
Chapter 74 *Page 182-183*
Chapter 75 *Page 184-185*
Chapter 76 *Page 186-187*
Chapter 77 *Page 188-189*
Chapter 78 *Page 190-191*
Chapter 79 *Page 192-193*
Chapter 80 *Page 194-195*
Chapter 81 *Page 196-197*
Chapter 82 *Page 198-199*
Chapter 83 *Page 200-201*
Chapter 84 *Page 202-203*
Chapter 85 *Page 204-205*
Chapter 86 *Page 206-207*
Chapter 87 *Page 208-209*
Chapter 88 *Page 210-211*
Chapter 89 *Page 212-213*
Chapter 90 *Page 214-215*
Chapter 91 *Page 216-217*
Chapter 92 *Page 218-219*
Chapter 93 *Page 220-221*
Chapter 94 *Page 222-223*
Chapter 95 *Page 224-225*
Chapter 96 *Page 226-227*
Chapter 97 *Page 228-229*
Chapter 98 *Page 230-231*
Chapter 99 *Page 232-233*
Chapter 100 *Page 234-235*
Chapter 101 *Page 236-237*
Chapter 102 *Page 238-239*
Chapter 103 *Page 240-241*
Chapter 104 *Page 242-243*
Chapter 105 *Page 244-245*
Chapter 106 *Page 246-247*
Chapter 107 *Page 248-249*
Chapter 108 *Page 250-251*
Chapter 109 *Page 252-253*
Chapter 110 *Page 254-255*
Chapter 111 *Page 256-257*
Chapter 112 *Page 258-259*
Chapter 113 *Page 260-261*
Chapter 114 *Page 262-263*
Chapter 115 *Page 264-265*
Chapter 116 *Page 266-267*
Chapter 117 *Page 268-269*
Chapter 118 *Page 270-271*
Chapter 119 *Page 272-273*
Chapter 120 *Page 274-275*
Chapter 121 *Page 276-277*
Chapter 122 *Page 278-279*
Chapter 123 *Page 280-281*
Chapter 124 *Page 282-283*
Chapter 125 *Page 284-285*
Chapter 126 *Page 286-287*
Chapter 127 *Page 288-289*
Chapter 128 *Page 290-291*
Chapter 129 *Page 292-293*
Chapter 130 *Page 294-295*
Chapter 131 *Page 296-297*
Chapter 132 *Page 298-299*
Chapter 133 *Page 300-301*
Chapter 134 *Page 302-303*
Chapter 135 *Page 304-305*
Chapter 136 *Page 306-307*
Chapter 137 *Page 308-309*
Chapter 138 *Page 310-311*
Chapter 139 *Page 312-313*
Chapter 140 *Page 314-315*
Chapter 141 *Page 316-317*
Chapter 142 *Page 318-319*
Chapter 143 *Page 320-321*
Chapter 144 *Page 322-323*
Chapter 145 *Page 324-325*
Chapter 146 *Page 326-327*
Chapter 147 *Page 32*

(b) Paper Docket Instance with Content Binding

Fig. 1. A Paper Docket

- *content providers* have to understand the utilization of their content so that they can decide whether and how they want to contribute;
- *content reviewers* need a clear image of their task so that they can judge the purpose of the entire action and the role of their contribution; and finally,
- *review evaluators* at the dean's office want to represent their interest by prescribing docket property types which are to be filled in by the reviewers.

In our example the department sees added value in publishing an up-to-date and extensive Research Report. Therefore, the dean's office periodically initiates a reviewing process by issuing docket instances and binding them to both existing content and to the responsible reviewers. Resource autonomy is maintained by attaching only a copy of last year's content.

Fig. 1(b) refers to an example of a paper docket *instance* created by enactment of a review docket based on the template of fig. 1(a). This docket instance initiates the reviewing of the currently published Research Report of the Software System Institute and finally reports the reviewing results back to the dean’s office. For this purpose, the docket instance establishes concrete bindings to the

- review content (*Attached please find a copy of ...*), and to the content provider (*Published by TUHH through version ...*);
- review results required by the dean's office (*...on External Projects ...*), and to the review evaluator (*From: Mrs. Amanda Miller; Phone: 3462*),

- review procedure (*provide update information ...*), and to the content reviewer (*To: Mr. John Smith*).

2.2 Docket Abstractions: Templates and Their Domains

Conceptually we define docket as a compound of three abstractions where one represents the document providing domain, a second one the domain of content reviewing, and the third one the domain of review evaluators and their value chains. Abstracting over traditional paper docket examples leads to a conceptual docket model with a common docket template structure as presented in fig. 2.

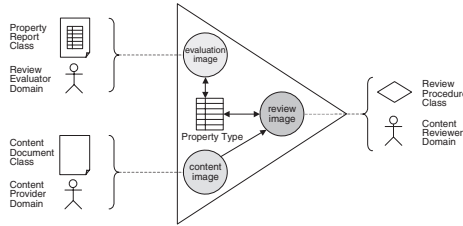


Fig. 2. Docket Templates and Domains

While the template level refers to domains, at the instance level concrete bindings to instances out of those domains are established and maintained: a content providing document is attached, a reviewing procedure instance is bound, and properties are specialized for content reviewing and review evaluation.

On the one side, docket *correlate* aspects of three different *domains*, on the other side, they *separate* the issues of three distinct *concerns* – docket design, presentation, implementation. It is essential to docket modeling to understand how both groups of tasks, domain correlation and concern separation, are distributed over the two *levels* of docket templates and docket instances. As a consequence, we concentrate on issues of docket design, and we discuss these issues on the two levels of docket templates and docket instances.

In this paper we leave implementation issues to a minimum. We want to emphasize, however, that one of the prime objectives of docket design is to enable a coherent and meaningful presentation (“best docket images”) of the three domain abstractions involved in each docket.

Content images on the docket template level inform potential content providers about the requirements expected by content instances at docket binding time. This might include type information as well as requirements on content-oriented quality, access rights, format, layout, space, time etc. The weaker requirements are based on copy semantics, reference semantics usually imposes more restrictions on content autonomy.

The example in fig. 1 shows that a substantial part of the docket is used to provide an image of what the review is all about. Though reviewers are au-

tonomous, the **review image** may impose specific constraints, e.g., by prescribing the order in which the content is to be read. Additionally, qualifications on the “career record” may be imposed on reviewers. Technical capabilities required by a performing reviewer can also be specified.

Evaluators are supposed to exploit the review results in the light of the encompassing value chain. They therefore need to communicate its relevant aspects by the **evaluation image**. This might include making additional aspects explicit: the evaluator’s identity and intent, the overall value chain, the identity of the docket initiator, etc. Evaluators express their demands on docket results essentially in terms of properties and their types. This is discussed in section 3.2.

3 Docket Enactment: Instances and Their Bindings

In the previous section we discussed docket templates and the consideration of the three domains involved. Fig. 3 depicts a docket instance with established bindings. Like type definitions in formal programming models, template information essentially serves the purpose to make domain assumptions explicit at docket design time and to assure that at docket enactment time only instances are bound which meet those assumptions.

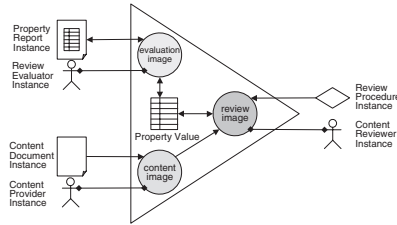


Fig. 3. Docket Instance and its Bindings

3.1 Docket Enactment

When a (step of a) value chain is to be carried out under docket control, an appropriate docket template has to be found or designed. From templates, instances are created, and a generic process scheme is invoked. As with workflow modeling we call this process scheme docket enactment. A generic docket enactment scheme is sketched by the UML activity diagram in fig. 4.

Binding docket performers starts with selecting individual entities and performers from each of the three domains. The initiative for docket enactment is often taken by the person performing as an evaluator. There may, however, be cases where the content provider, the reviewer, or a third party initiates docket

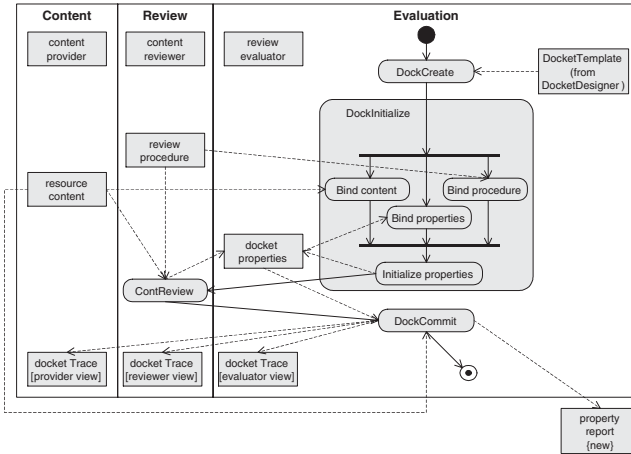


Fig. 4. Generic Docket Enactment Scheme

enactment. The selected domain entities are checked against the three corresponding images defined by the docket template. One performer might base his decision about taking part on the choice of the others (not shown in fig. 4).

By docket binding, the content is provided, the properties are initialized, and the review procedure is started. In general, each domain contributes a static part provided by the image, and a dynamic part provided via performer binding.

Interpreting docket modeling as a contracting activity, docket enactment can be seen as the signing of a contract. Docket images should enable all three performers to understand the overall purpose of the docket which, finally, all three have to agree and to accept. This may require further negotiation. Once the performers are bound to the docket instance they are bound by the “contract” [28].

3.2 Docket Properties and Traces

Properties serve as the basis for defining and achieving the added value through a docket: on the template level, properties and their evaluation image context communicate the goal of the review, on the instance level, properties take up the review results. After a docket is terminated properties are stored in a *property* or *review report*. This is all the evaluator gets out of a docket enactment.

We refer to property specification on the template level as the property type. It includes the structure of the properties, the range of possible values, the units and metrics in which to measure them, constraints on the values’ accuracy, tolerance etc. In the most general case of human reviewers, a property specification might be given in natural language (compare fig. 1). In the most specific case, a type definition for some software component will be provided.

By defining and setting properties, the evaluator influences the review process. The values of the initialized properties are input for the reviewer additional to the property type. Under extended protocols, the evaluator may observe the

reviewer at work. If the review is not likely to satisfy the evaluator's expectation, he may adjust the properties to guide the course of the review.

As a value container for results, the properties are implemented by some data or document model. In our docket framework (see section 6) we experiment with SGML and XML [11], exploiting the possibility of mixing dynamic and static content in a defined structure. An important advantage is the possibility to provide parts of documents as valid documents; for XML "fragment interchange" is currently being discussed [8]. Advanced property models will require extensions for typed attributes with metrics, tolerances, taxonomy extension etc. [1].

Often the relationship between the property structure and those parts of the document which provide the content has to be preserved. If the content structure is known to the reviewer and can be encoded in the properties, the mapping can be implemented by cross-references. In our docket framework, both properties and contents are given as documents with known DTD, and we use links defined by the XML Linking Language for cross-referencing [15]. If direct references from one model to the other are not possible, annotation models might be appropriate [27]. In such cases properties are annotations that are attached to those parts of the content which they comment.

At docket commit time, any binding between the three docket contributions is released. However, since docket enactment is a value in itself to any enterprise as well as to the performers, each enactment is documented. The three performers can expect at least two benefits from such a documentation. First, it serves as a receipt of their participation in some docket enactment, be it for controlling purposes or for proving that they fulfilled the contract defined by the docket. Second, they might use it as a measure of their own professional performance.

In our docket framework each enactment produces a trace. We experiment with various trace and traceability models from software engineering [23], requirements engineering [25,5], and workflow history management [14]. Our approach to traces is property-based, i.e., they are in essence versioned and materialized views on the properties and the review procedure. For a detailed discussion of the relationship between document, version, and process models see [12].

For added organizational value, traces are stored in a database [22] maintained by the docket framework. It allows queries on the trace extent which provide a basis for data warehousing over content-oriented value chains.

Each docket performer may get a specific variant of the trace and may use it at will – with one notable exception. To serve as a proof in the other performers' eyes, traces have to be immutable. This is the basis for acceptance in a legal context or as a measure for the reviewer's professional performance.

4 Extended Docket Scenarios

Our basic docket model presented so far is based on the three principles: resource protection by controlled access, content review by reviewer assignment, and evaluation through property-based value chains. Based on our first experi-

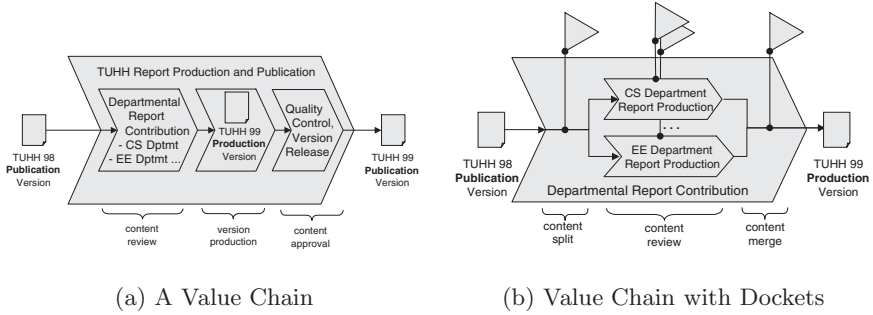


Fig. 5. Value Chains in the Research Report

ence with docket-driven application development we demonstrate how the basic docket model can be used to support more general content management schemes.

Patterns of particular interest are those which span over multiple dockets. They are required, for example, in cases where results of some initial review are used to create new document versions.

The docket from the example of fig. 1 aims at producing the annual version of the CS department’s part of the of the Research Report. The overall value chain is modeled by fig. 5(a): it updates the 1998 document version and publishes the 1999 edition. For further discussion, we refer to the “fish” notation (or “Chevron” symbol) which is also used in SAP’s reference model [6] as an informal mean to name, structure and relate value chains.

Value chains are composed recursively. Fig. 5(a), for example, shows three sub-chains. Starting point of the value chain is the published version of the Research Report. The Production Version constitutes an intermediate state in our value chain, the Publication Version is its final result.

Value chain refinement reaches the docket level as soon as added value can be implemented by routine review and evaluation of content. Fig. 5(b) shows a refinement of the first member of the value chain into parallel activities. It is assumed that the deans’ offices know how to implement their value chain by routine reviewing of existing contents. In fig. 5(b) this is symbolized by “raised docket flags”. This connects the example back to our initial discussion (see fig. 1).

Two other parts of the value chain in fig. 5(b) are candidates for dockets, content split and merge. Research Report structure and organizational structure are fairly isomorphic, as to be expected with universities and their departments, so they can be carried out by dockets with little or no human intervention.

So far, our example concentrates on dockets for report reviewing which produces properties and leaves the initial content unchanged. The central step in the value chain which produces a report update version (see fig. 6(a)) has not yet been addressed on the docket level. When designing update dockets for version production one observes an interesting inversion of roles between property

and content. As indicated by fig.6(b), the document by which the *content* for an update docket is provided, is produced and probably owned as a *property* report by some reviewing step further up in the value chain.

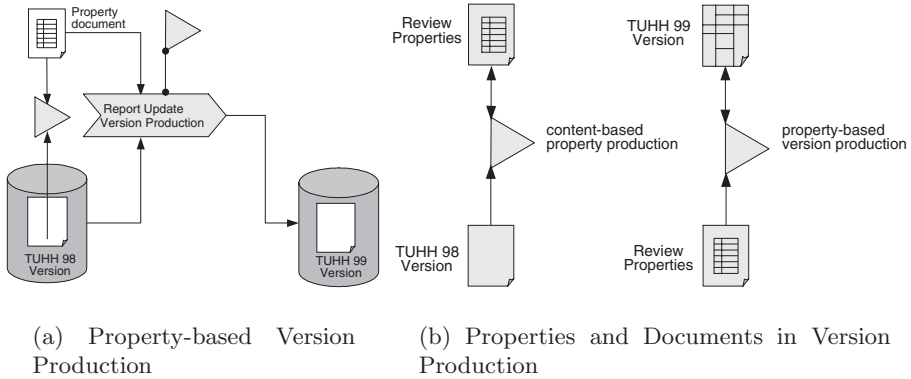


Fig. 6. Dockets for Version Update

The deeper reason for making version update a two-step, three-state value chain lies in the fact that in professional content management one has to distinguish carefully between the three cases of ownership, on one hand – initial content, update information, update version – and, on the other, the two value-adding processes and their responsibilities – content reviewing and update.

5 Docket Lifecycle

There are clearly distinct phases which determine the lifecycle of a docket. The first and most important phase is problem analysis which is based on content-oriented use cases and value chains; this phase results in the design of a docket template. Two additional phases are the phases which deal with docket presentation and docket implementation.

5.1 Docket Design

Besides rather generic patterns which are shared by most dockets, e.g., patterns for docket enactment (compare fig. 4, section 3.1), there are many more specific decisions to be made by the docket designer: specification of review procedures, property sets and access protocols through which content is provided.

Fig. 7 specifies as part of a docket design the typical tasks to be performed by a reviewer. In this case a reviewer performs his task in one go, i.e., without any interaction with the provider or the evaluator. The reviewer begins by taking

notice of the content (**Read Content**). In fig. 7 we distinguish three alternative review procedures: The first kind expects the reviewer to just read the content (**notification**). For the second kind of task (**ranking**) the reviewer is supposed to produce a property by selecting a value from a discrete value set as some ranking. For the third kind of review procedure (**annotation**) properties are expected to be of a type which leaves room for the reviewer's creativity, e.g. free text, images, speech etc. Any review ends with acknowledging that the review was performed.

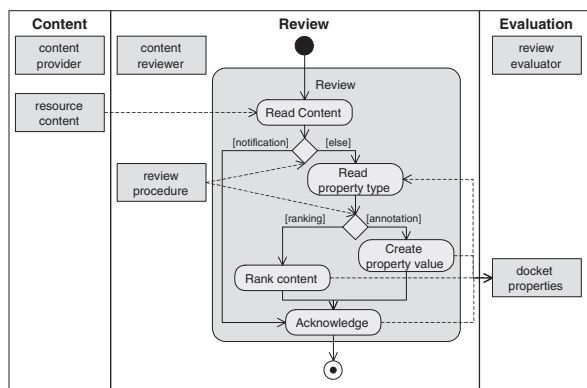


Fig. 7. Activity Diagram of Standard Review Image

5.2 Docket Presentation and Implementation

Good docket *presentation* is essential to the entire approach. Its objective is to communicate by its three images the objectives of the docket at hand. The partitioning into three domains which is made by the model is given up in favor of a coherent docket presentation. Our paper dockets in fig. 1 already show such coherence, the example subsequently discussed demonstrates some of the additional potential of screen-based docket presentation.

Our view on dockets is heavily influenced by experience with a digital library project with our colleagues from the Art History department at Hamburg University. The project Warburg Electronic Library (WEL) [30] deals with objects of art which are or have been used for conveying political concepts, messages and goals. In the WEL project we distinguish carefully between various signifier-signified relationships such as iconic, indexical and symbolic references [7].

The WEL provides docket-driven applications for icon classification, annotation, tracking etc. Fig. 8 shows the presentation of a WEL docket. The left-hand-side offers a selection of the ontology terms used by the WEL for image classification, the lower right-hand-side a stack of images, and the upper right-hand-side a text which specifies the classification task. The images are the content for which added-value is to be produced. Additional content is provided by a file named in

the task description. This text together with the drag-and-drop layout elements constitutes the image of the review procedure. On the template level, docket properties are represented by the various fields, e.g., title (“Titel”), artist name (“Künstler”), keywords (“Schlagwort”, “Stichwort”), and their types. The concept hierarchy is also part of the property type. On the instance level, properties are partially initialized (e.g., keyword “mit Tier” or artist “(Cesare Ripa)”).

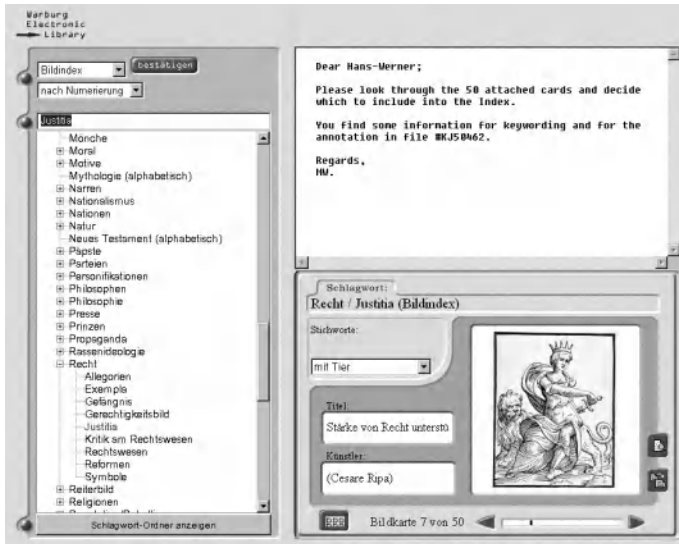


Fig. 8. Docket for Image Classification: Presentation Level

The university Research Report discussed in section 4 is another docket-driven application project. The paper-based procedure of updating the report has been replaced by an online system where the university owns and provides the content, the departments organize the production of new document versions, and the institutes review reports regularly and provide update information.

Besides driving docket presentation, docket design also captures the essential requirement for docket *implementation*. Since this paper concentrates on the conceptual modeling aspects of dockets only a few references to our implementation experience will be made. Some requirements have already been discussed above in the context of generic docket activities (compare figs. 4, 7). Our first insight into an emerging docket framework is outlined in the subsequent chapter.

6 Towards a Docket Framework

Content management patterns as discussed in sections 3 and 4 are becoming common in online content production and publishing. As a result we present the

architecture of an emerging docket framework and discuss some of its components. We built upon CoreMedia, a commercial product for editorial management developed by a Hamburg-based software vendor. Work on a docket framework is considered to be a contribution towards an emerging *Docket Technology*.

For resource autonomy reasons, content providers and docket handlers (editors, producers, publishers) reside on separate servers, the former on a publication server, the latter on a production server. Demands for docket cooperation lead to the following architecture (fig. 9):

- *publication servers* present, store and protect released content;
- *production servers* have controlled access to publications and host docket;
- *specific docket* manage trusted content exchange between the servers;
- *generic docket* communicate and encapsulate value-adding services.

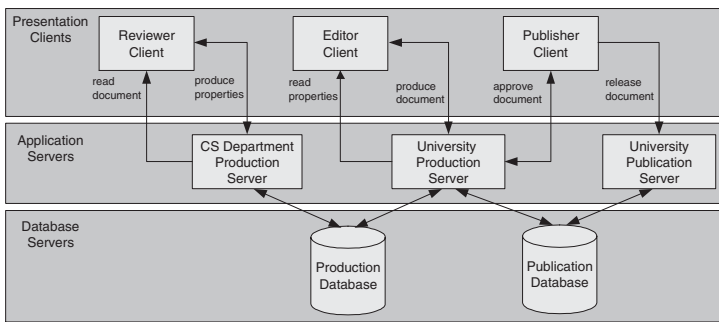


Fig. 9. System Architecture for Online Report Production and Publication

By decoupling the two servers and their repositories, content protection and resource autonomy are established. Specific docket templates for content split and content merge are designed to control content exchange between production and publication servers and to minimize the need for human intervention.

Dockets for content reviewing tasks are represented on editor clients while docket for the more mechanical parts of content management such as document split, merge, compilation etc. reside on producer clients. Both kinds of docket share the production server which interacts with the underlying production database. Our docket framework supports integrity management and type checking on various levels and offers a range of binding alternatives [29].

Docket presentation clients are currently implemented in two versions: the one version is a Java-based client program, the other one is a web-based, form-oriented version implemented by HTML pages and Java servlets. The specifications of the review procedure are visualized, and review operation is monitored.

The development of a generic client which can generate initial presentations based on the images of docket templates is under way. Together with the growing generic functionality provided by our docket framework, support for docket visualization will substantially speed up docket-based application development.

7 Related Work

Throughout this paper we already mentioned some links to other models. In this section we have a closer look on their relation to dockets.

7.1 Workflow Models

Both dockets and workflows are based on models and methodologies which enable enterprises to find abstract descriptions of their essential value chains, see commonalities between processes, and benefit from such insight. Both models profit from the fact that stereotypic processes can be generalized by templates.

Generic workflow models focus on the flow of control by which processes essential to the enterprise are activated and constrained. Such processes are usually distributed over organizations and require validation, optimization and control [10]. In contrast, dockets do *not* aim at representing entire value chains.

Potential docket performers have to be able to decide whether and how to participate in docket enactment. For this purpose they have to understand the value chain behind a docket. Most workflow models have shortcomings in this respect. The Toronto school with its conceptual modeling background is one of the few exceptions: the i^* method [32], for example, pays specific attention to capturing the intent behind an enterprise's activity.

Some approaches include resource models and control instances by integrating organizational models with workflow and data models [3]. The organizational model defines the possible delegation of work. In docket modeling we assume, however, that docket performers are autonomous actors which may not even belong to the same organization.

Electronic Circulation Folders (ECFs) [24] offer a metaphor similar to dockets. They are used to clarify migration, ownership, and responsibility. ECFs address users who are comparable to reviewers, not different domains like dockets do. The main goal of ECFs is asynchronous cooperation as we address it on the value chain level. Resources are not of interest in their own.

7.2 Agents

Performers who cooperate in a docket scenario can be seen as independent agents bound by a common task to be solved. They are in a multi-agent setting observing each others behavior. An evaluator, for example, may observe a reviewer to make sure that the review will be of the quality the evaluator requires. Requirements for agents in improvising scenarios as discussed by [9] also hold for dockets: Their shared goals are weakly defined, leaving much space for creativity, and the constituents (agents) have to work together closely.

7.3 Speech-Act-Based Cooperation

As pointed out earlier, due to performers' autonomy, docket design requires an agreement on the conditions of the collaboration. One approach to model

such negotiations is based on the speech act theory, which has already been applied successfully to workflow modeling [31]. Implementations of speech act models are the Action Workflow model [21] and the Business Conversations [17]. For Business Conversations an agent framework [20] has been developed which stresses the autonomy of the actors.

A recent draft of the Internet Engineering Task Force discusses the issue of finding an agreement for data format to be used in communications [13].

7.4 Annotation Models

The structural requirements of document models have already been discussed in section 3.2. An annotation model dealing with these requirements is currently under investigation by [4]. Two specific kinds of annotations are of particular interest to dockets: seals of approval (SOAPs) and usage indicators [26]. SOAPs are used by experts to give ratings to documents. Reviewing with dockets based on enumeration type properties is similar to SOAP assignment. Usage indicators measure a document's value by showing how often it has been read and possibly by whom. Thus, they are similar in purpose as traces produced by dockets.

8 Concluding Remarks

In this paper we give a first presentation of dockets as a conceptual model. We emphasize the potential of capturing the essence of routine processes by generic docket templates and stress the importance of docket design and presentation.

Besides extending the functionality of our docket framework we are particularly interested in further developing our library of docket templates. Thus, we intend to capture in a single conceptual model generic patterns essential for content management and content-based service development.

Our previous work in highly generic computational models with persistence and migration support [16,18,19] turns out to be of particular importance for the development of generic and high-level docket technology.

Acknowledgements

Special credit is given to Higher-Order GmbH and some of its customers for numerous discussions of conceptual and technical issues in content management.

Furthermore, we want to thank our colleagues and fellow researchers of the WEL project (Grant by Hamburg State) and the members of the EU-Canada collaboration on Cooperative Information Systems (Grant by Esprit under project ISC-CAN-080 CIS), in particular to John Mylopoulos, Matthias Jarke, and Eric Yu. Credit is also given to Misha Missikoff, Bernd Thalheim and Florian Matthes for continuous constructive contributions to our research.

Finally, special thanks go to our friends, Diane and John Smith from Xerox Corp., for their very valuable input from the perspective of professional document management and presentation.

References

1. Biron, P.V., Malhotra, A. (eds.): XML Schema Part 2: Datatypes. World Wide Web Consortium Working Draft (1999) <http://www.w3.org/TR/xmlschema-2> 254
2. Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual Modeling. Springer (1984) 249
3. Bucu, M.J., Chaar, J.K.: On the Integration of Project, Workflow, and Document Management. Technical Report, IBM (1996) 260
4. Buneman, P., Deutsch, A., Tan, W.: A Deterministic Model for Semistructured Data. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats (1998) 261
5. Communications of the ACM, Special Issue on Traceability. 12 (1998) 254
6. Curran, T., Keller, G., Ladd, A.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Prentice Hall (1998) 255
7. Deacon, T.W.: The Symbolic Species. W.W. Norton & Co. (1997) 257
8. Grosso P., Veillard, D. (eds.): XML Fragment Interchange. W3C Working Draft (1999) <http://www.w3.org/TR/WD-xml-fragment> 254
9. Hayes-Roth, B., Brownston, L.: Multi-Agent Collaboration in Directed Improvisation. Technical Report, Stanford University (1994) 260
10. Hollingsworth, D.: The Workflow Reference Model. Workflow Management Coalition (1995) <http://www.aiim.org/wfmc/standards/docs/tc003v11.pdf> 260
11. Jelliffe, R., Jelliffe, R.A.: The XML and SGML Cookbook: Recipes for Structured Information. Prentice Hall (1998) 254
12. Joeris, G.: Change Management Needs Integrated Process and Configuration Management. ESEC/SIGSOFT FSE (1997) 125-141 254
13. Klyne, G.: Protocol-Independent Content Negotiation Framework. Draft, Internet Engineering Task Force <http://info.internet.isi.edu:80/0/in-drafts/files/draft-ietf-conneg-requirements-02.txt> 261
14. Koksall, P., Arpinar, S.N., Dogac, A.: Workflow History Management. SIGMOD Record 1 (1998) 67-75 254
15. Maler, E., DeRose, S. (eds.): XML Linking Language (XLink) W3C Working Draft (1998) <http://www.w3.org/TR/WD-xlink> 254
16. Mathiske, B., Matthes, F., Schmidt, J.W.: On Migrating Threads. NGITS 1995 261
17. Matthes, F.: Business Conversations: A High-level System Model for Agent Coordination. DBPL 1997: 355-372 261
18. Matthes, F., Schmidt, J.W.: Persistent Threads. VLDB 1994: 403-414 261
19. Matthes, F., Schröder, G., Schmidt, J.W.: Tycoon: A Scalable and Interoperable Persistent System Environment. In: M.P. Atkinson (ed.): Fully Integrated Data Environments. Springer-Verlag (1999, in press) 261
20. Matthes, F., Wegner, H., Hupe, P.: A Process-Oriented Approach to Software Component Definition. CAiSE 1999: 26-40 261
21. Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: The ActionWorkflow Approach to Workflow Management Technology. CSCW 1992 261
22. Muth, P., Weissenfels, J., Gillmann, M., Weikum, G.: Workflow History Management in Virtual Enterprises Using a Light-Weight Workflow Management System. RIDE 1999: 148-155 254
23. Palmer, J.D.: Traceability. In: Dorfman M., Thayer, R. H. (eds.): Software Engineering. IEEE Computer Society Press (1997) 254

24. Prinz, W., Kolvenbach, S.: Support for Workflows in a Ministerial Environment. CSCW 1996: 199-208 **260**
25. Ramesh B., Jarke M.: Towards Reference Models for Requirements Traceability. To appear in: TSE (1999) **254**
26. Röscheisen, M., Mogensen, C., Winograd, T.: Shared Web Annotations As A Platform for Third-Party Value-Added Information Providers: Architecture, Protocols, and Usage Examples. Technical Report (1994)
<http://www-diglib.stanford.edu/diglib/pub/reports/commentor.html> **261**
27. Röscheisen, M., Winograd, T., Paepcke A.: Content Ratings and Other Third-Party Value-Added Information – Defining an Enabling Platform. D-Lib Magazine (1995) <http://mirrored.ukoln.ac.uk/lis-journals/dlib/dlib/dlib/august95/stanford/08roscheisen.html> **254**
28. Ryu, Y.U.: Specification of Contractual Obligations in Formal Business Communication. DKE 26(3): 309-326 (1998) **253**
29. Schmidt, J.W., Schröder, G., Niederée, C., Matthes, F.: Linguistic and Architectural Requirements for Personalized Digital Libraries. Int. J. Digit. Libr. **1** (1997) 89-104 **259**
30. Warburg Electronic Library, <http://www.sts.tu-harburg.de/projects/WEL> **257**
31. Winograd, T.: A Language/Action Perspective on the Design of Cooperative Work. Human-Computer Interaction **1** (1987-88) 3-30 **261**
32. Yu, E.S.K., Mylopoulos, J.: From E-R to “A-R” – Modelling Strategic Actor Relationships for Business Process Reengineering. ER 1994: 548-565 **260**

Workflow Specification in TRAMs

Markus Kradolfer*, Andreas Geppert, and Klaus R. Dittrich

Department of Computer Science

University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland

{kradolf,geppert,dittrich}@ifi.unizh.ch

Abstract. Adequate methods to workflow system design allow for all relevant aspects of workflows, such as organizational, structural, and behavioral aspects. Additionally, since in most organizations basic components of workflow systems already exist, workflow system development should support a component-oriented way of integrating existing parts into (new) workflow systems.

In this paper, we present the TRAMs-approach towards workflow specification. This approach supports the specification of the aforementioned functional aspects of workflows. The basic modeling construct in TRAMs are workflow types, which can either be complex or atomic (activity types). Complex workflow types define a set of subworkflows, and control and data flows among the subworkflows. Organizational entities and relationships can be defined and related to activities through task assignment rules. TRAMs-specifications are modular, and their parts are therefore more likely to be reusable in other specifications. Finally, specifications are activity-centric (in contrast to state-centric approaches) which allows a more intuitive way of modeling.

1 Introduction

Workflow management has recently found great attention in the area of information systems, as it allows to capture knowledge about business processes, to define workflows in a formal language/framework, and to enact workflows according to their specification. A *workflow specification* defines the structure of workflows (e.g., atomic activities/steps), processing entities responsible/capable of executing these activities, and further constraints such as execution or temporal dependencies. *Workflow management systems* (WFMS) [10,13] are the software systems that support the specification and the execution of workflows. A WFMS together with a set of workflow specifications and the processing entities is called a *workflow system*.

In most approaches, workflow systems are developed in a top-down manner in which, starting from real-world processes, workflows are specified. In this way, activities included in the workflow are identified and resources (i.e., systems or humans) are assigned to these activities. Often, the specification of activities in

* The work of Markus Kradolfer has been supported by the Swiss National Science Foundation (grant 2000-046925).

a workflow also requires rules which determine when and under which conditions they can/must be executed. Thus, workflow systems are developed in a rather ad-hoc manner, without systematic consideration of information about already defined activities, workflows, and organizational aspects.

In contrast to this top-down approach, we argue that workflow systems should be component-based and developed using a spiral model [2]. In this way, workflow system development starts with a set of activity and/or already existing workflow definitions that are aggregated into more complex workflow definitions. Thus, a workflow system (and consequently, the processes defining how organizations conduct their business) is obtained by putting together artifacts of different types, such as activities, workflows, organizational entities and relationships. In order to enable such a component-based approach to workflow systems, several prerequisites are crucial:

Abstraction and modularity The pluggable artifacts must be represented at an adequate level of abstraction. This means that “activities” are the basic, atomic elements of workflows, whose structure and implementation is not required to be known. Moreover, modularity must be maintained, meaning that artifacts can be specified in a self-contained way. Artifacts which are only related in the context of specific workflows must be defined independently, and are not allowed to make assumptions about the context they will be used in.

Workflow composition In order to develop workflows by composition, it must be possible to connect workflow system artifacts in flexible ways. Based on the principle of abstraction and modularity, only the results of (sub)workflows are of interest, but not the way how they have been achieved. In this way, an activity-centric view of workflows is possible, which focuses solely on how workflows are composed into more complex ones.

Flexible organizational modeling A basic requirement for workflow specification approaches is support for the specification of the assignment of activities to processing entities. To support the specification of activity assignments based on existing organizational relationships, the modeling of organizational aspects must be supported. As pointed out in [3], it is not sufficient, as done in most existing approaches, to provide a fixed set of entity types (e.g., only “processing entity” and “role”) and relationship types (e.g. “can play role”, “supervisor of”, and “reports to”), since the organizations to be modeled often differ significantly.

Flexible execution constraint specification Certain aspects of a workflow, such as the exact execution order of the steps to be carried out, might not be known before execution time and therefore cannot be specified in advance. Workflow specification must allow for flexibility in those cases where precise execution constraints can be determined only at execution time.

Failure handling and recovery During workflow execution several kinds of failures, including system as well as semantic failures, may occur. Since workflows usually are of long duration and considerable amounts of work are done during their execution, it is not acceptable that a failure causes the

complete rollback of workflows. Rather, forward and/or partial backward recovery strategies should be employed which might require the definition of how workflows can be compensated as well as the definition of alternative execution paths.

Explicit representation of the development history For practical cases, it is necessary to keep track of the development history of workflow systems as well as of variants of workflow types. It is thus not sufficient to maintain only a single version—namely the current one—of workflow types.

The contribution of the TRAMs workflow specification approach is that it meets all these requirements, in addition to the purely functional requirements (such as workflow structure specification). Abstraction and modularity is achieved by separating the concerns of the various specification elements (activities/workflows, organizational facts, execution constraints). Furthermore, in TRAMs workflow execution results can be referred to through the notion of *end state*, whereby an arbitrary number of such states can be defined. The state transition graph, however, is fixed for workflows. Flexible organizational modeling is supported, whereby new organizational entity as well as relationship types can be defined if needed. Execution constraints such as data flow and control flow are separated from each other and from activities/subworkflows they refer to. It is thus possible to reuse activities and to assign new constraints to them in new contexts. TRAMs allows reactions to failures to be defined within workflow specifications, but a separate modeling of processes and exception handling, as proposed e.g. in [12], is not supported. The development history is represented explicitly via the notion of workflow type versions¹. This version model can express revisions as well as variants of workflow types.

The remainder of the paper is organized as follows. Chapter 2 introduces the terminology used throughout the paper. An example workflow type is presented in Chap. 3. Chapter 4 defines the organizational and Chap. 5 the workflow model. Chapter 6 discusses related work, and Chap. 7 concludes the paper.

2 Terminology

A *workflow* is either a basic work step, called *activity*, or a *complex workflow* that consists of one or more other workflows. A workflow x that is contained in another workflow y is called *subworkflow* of y and workflow y is called *superworkflow* of x . Workflows that have the same superworkflow are called *sibling workflows*. A workflow that has no superworkflows is called *top-level workflow*.

Activities are carried out by *processing entities* (PE) which can range from humans to arbitrary software systems. Each activity and complex workflow is an instance of a certain *workflow type*. More precisely, an activity is an instance of an *activity type* and a complex workflow is an instance of a *complex workflow type*. From a workflow type, any number of instances (workflows) can be created.

¹ Details about workflow type versioning in TRAMs are omitted in this paper due to space restrictions but can be found in [14].

A workflow type definition (synonymous to workflow specification) is a formal description of the common aspects of a set of workflows. The collection of the workflow types defined at a certain point in time forms the *workflow schema*.

A WFMS is a software system that supports the specification, enactment and monitoring of workflows. Workflow enactment does not mean that the WFMS itself carries out the activities workflows consist of. Rather, the WFMS ensures that the processing entities execute the activities in a coordinated way according to the corresponding workflow specification. A WFMS consists of a specification component and an enactment component. The *specification component* supports the specification, persistent storage and management of information related to workflow types and processing entities. The *enactment component* is responsible for the enactment of workflows according to the workflow specifications created with the specification component.

The specification component offers one or more *specification languages*, which are formal languages that offer constructs to capture the information necessary to enact workflows. A specification language has an underlying *model*, which defines the modeling concepts the language offers through its constructs in a high-level way.

In this paper, we define a workflow model and an organizational model: the *workflow model* defines a set of modeling constructs for the specification of workflows, and the *organizational model* defines a set of modeling constructs for the specification of processing entities, their properties, and the relationships that exist among them.

3 Example Workflow Type: Health Insurance Claim Processing

In this chapter, we present an example workflow type. The application scenario is the processing of a health insurance claim (HIC) in an insurance company (see Fig. 1). While we give only an informal overview of the workflow type here, Chap. 5 defines various aspects of the workflow type in greater detail and in a more formal way.

Once a HIC is received, a clerk creates an electronic dossier containing the diagnosis, the treatments, and the costs, and a reference to the insurance policy

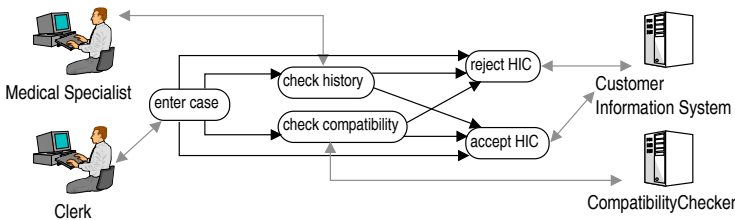


Fig. 1. Sample workflow: health insurance claim processing

in the insurance company database (**enter case**). If the cost is 150 Swiss Francs or less, the HIC is rejected (**reject HIC**), i.e. a notification of the rejection is printed. Otherwise, if the cost is between 150 and 500 Swiss Francs, the HIC is accepted (**accept HIC**), i.e. an entry in the payment history of the applicant is made and a payment check is printed. **accept HIC** as well as **reject HIC** are executed by a software system called **Customer Information System**. If the cost is 500 Swiss Francs or above, two checks are done in parallel. An expert system named **CompatibilityChecker** validates whether the treatment suits the diagnosis (**check compatibility**) and a medical specialist checks if the treatment makes sense with respect to the patient's history (**check history**). If one or both of these two checks fail, the HIC is rejected (**reject HIC**). Otherwise, the HIC is accepted (**accept HIC**). Finally, a law specifies that the insurance company has to react (either positively or negatively) at the latest after six weeks. Thus, we assume that if the HIC has not been rejected or accepted after six weeks, the claim is automatically accepted.

4 The Organizational Model

As mentioned above, a flexible organizational model is indispensable for successful workflow system development. In this section, we describe the organizational model of TRAMs, which forms the basis of the organizational perspective of workflow system designs (see Sect. 5.4). An organization is viewed as a set of *organizational entities* (such as processing entities, roles, groups, and departments), and a set of *organizational relationships* that exist among the organizational entities (e.g. the “supervisor of” relationship between two processing entities). An organization is specified by an organizational schema and an organizational database. The *organizational schema* consists of a set of organizational entity and relationship types. The *organizational database* comprises organizational entities and relationships that are instances of the types contained in the organizational schema.

An *organizational entity type* possesses a name and a set of attributes, where an attribute definition is a pair (attribute name, attribute type). The supported attribute types are those definable in the Object Database Management Group Object Definition Language (ODMG ODL) [5]. An *organizational relationship type* specifies a name and associates a set of organizational entity types. Since the kinds of entities and relationships may differ considerably from organization to organization, the organizational schema is not fixed and the specification of user-defined organizational entity and relationship types is supported. For instance, it might be required to express specific abilities of PE; in this case, one would define an entity type named **capability** and a relationship type **has_capability** that associates PE (processing entities) with capabilities.

Three organizational entity types and two organizational relationship types are predefined since they are found in many organizations. The predefined organizational entity types are **processing entity**, **role**, and **organizational unit**. A processing entity represents an individual “resource” (i.e., a human or a sys-

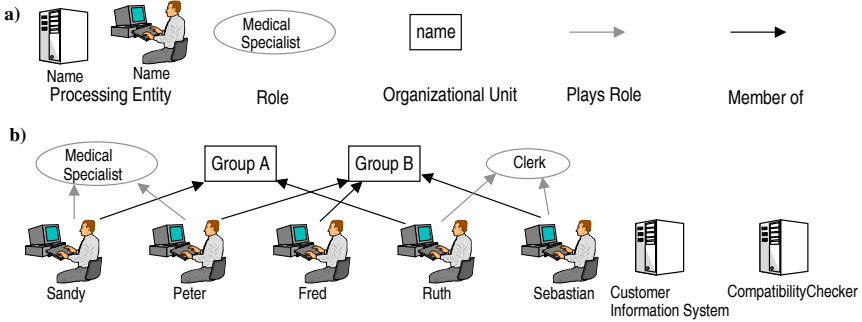


Fig. 2. Organizational database for the HIC processing example

tem). Roles model a specific contexts in which PE act (e.g., a PE can act as a project leader). Organizational units represent collections of PE such as departments, groups, and projects. All three predefined organizational entity types define an attribute `name`. In addition to the predefined organizational entity types, there are two predefined organizational relationship types `plays_role` and `member_of`. A `plays_role` relationship associates a PE with a role it can play. A `member_of` relationship relates a PE with an organizational unit.

It is important to note that the organizational database may change over time. This means that, e.g., processing entities are added to/removed from the database or that processing entities may loose or gain their membership in organizational units in the course of time.

A design tool for the TRAMs organizational model has been developed. This tool allows to create and persistently store organizational schemas and databases.

Graphical representations of the predefined constructs proposed for the organizational model are shown in Fig. 2a (user-defined constructs are represented textually). Figure 2b presents an organizational database for the HICProcessing example of Chap. 3 using the predefined organizational entity and relationship types.

5 The Workflow Model

In order to adequately define workflows, a workflow model is needed that covers the following perspectives (see [6]):

- functional/structural perspective: *Which* workflows have to be performed and how are they structured?
- informational perspective: Which *data* is produced and consumed by workflows, and how do data flow among workflows (*data flow*)?
- behavioral perspective: *When* have workflows to be performed (control flow)?
- organizational perspective: *Who* has to carry out the activities?

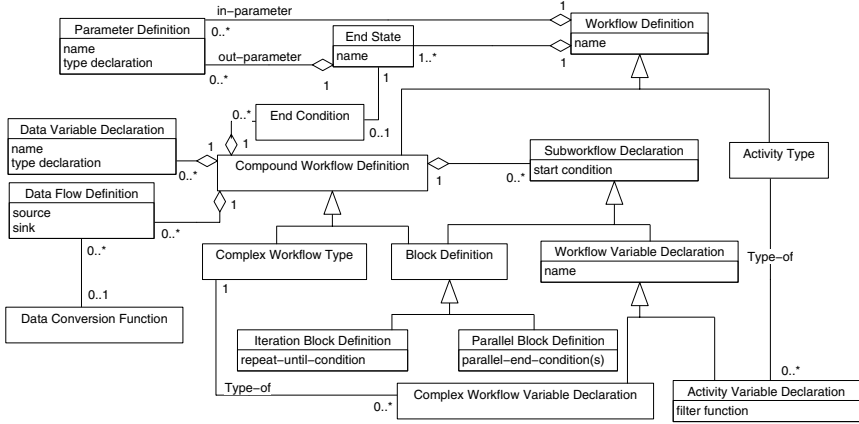


Fig. 3. Overview of the TRAMs workflow modeling constructs

In this section, we introduce the workflow model of TRAMs (see Fig. 3 for an overview of all constructs in UML-notation).

Each perspective is covered in a separate section (Sects. 5.1 through 5.4); each section contains a discussion of the respective modeling constructs, graphical representations, and an illustration using the running example.

Based on the workflow model described in this paper, both a concrete textual language and an editor have been developed. The editor implements creation and modification of workflow specifications using textual and graphical elements. It also supports the persistent storage and retrieval of workflow specifications.

5.1 Functional/ Structural Perspective

An important aspect of the workflow model is to provide for the adequate modeling of the workflow structures (i.e., the aggregation of basic workflows or activities into more complex ones). This aspect is covered by the *structural perspective*.

The central construct in the workflow model is the *workflow definition* (WFD) that specifies either an *activity type*, representing a basic work step, or a *compound workflow definition*. A compound workflow definition is either a *block definition* or a *complex workflow type*. A block definition structures parts of a compound workflow definition, without in turn being a full-fledged workflow type.

Both block definitions and complex workflow types define a set of subworkflow declarations. A *subworkflow declaration* is either a *workflow variable declaration* or a block definition. A workflow variable declaration consists of a variable name and a reference to a workflow type; it serves to incorporate this type into the compound workflow. The variable name is needed to unambiguously refer to the subworkflow in conditions and data flows, especially because it is possible

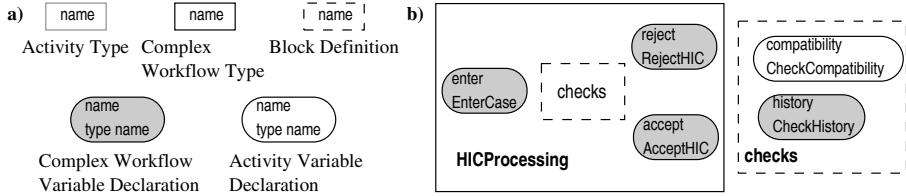


Fig. 4. Structural/functional perspective: graphical notation and sample usage

to include the same workflow type multiple times within a compound workflow definition.

The block definition construct is supported in order to provide the possibility to locally structure workflow definitions and to specify the sequential or parallel execution of a variable number of workflows. A (normal) block definition, an iteration block definition and a parallel block definition all include the specification of a set of subworkflows (blocks and workflow variables). The meaning of a parallel block is that its subworkflows are executed in parallel a variable number of times (where the concrete number of parallel subworkflow instances can only be determined at runtime, but not at specification time). Likewise, an iteration block executes its subworkflows repeatedly, where again the number of iterations can only be determined at runtime.

Figure 4a shows the graphical representation of the constructs introduced in this section. Figure 4b shows the structural perspective of the workflow type **HICProcessing** using the graphical elements defined in Fig. 4a.

5.2 Informational Perspective

In the informational perspective, the focus is on the data consumed and produced by workflows and on the data flow among workflows. It is essential that all the data flows among workflows are explicitly specified. Otherwise, if workflows implicitly exchange data (e.g., via a commonly accessed database) the reuse of existing workflow types is complicated, if at all possible.

In TRAMs, the informational perspective allows to model parameters of workflows, local variables of workflows, and data flow.

Parameters For each complex workflow type, activity type and block definition, in(put) and out(put) parameters can be defined. The parameters are defined using ODL. In-(out-) parameters model data consumed (produced) by a workflow. Each out-parameter of a workflow is associated to exactly one end state of the workflow (see Sect. 5.3 for details about end states). As already mentioned, the subworkflows that are defined by a parallel block are executed a variable number of times, where each set of subworkflows is called *parallel set*. Each parallel set needs a set of in-parameter values and produces a set of out-parameter values. Therefore, parallel blocks have

exactly one in-parameter (called `in_parameter`), and one out-parameter (called `out_parameter`) per end state. An `in_parameter` (`out_parameter`) is of ODL type set, whose elements are ODL structures defining a set of in-parameters (out-parameters) for a parallel set.

At the time a parallel block is started, for each element in `in_parameter` a parallel set is started. Thereafter, for each parallel set an element in `out_parameter` of the end state `end_z` is produced at the time the parallel block ends in `end_z`.

Data Variables Data variables serve as intermediate stores of data produced and consumed by workflows. Data variables can be defined by compound WFD (complex workflow types and block definitions). The scope of a data variable is local to the compound WFD which defines it.

Data Flows A data flow definition relates a *data source* to a *data sink*. A data source or sink is either a parameter or a data variable. Data flow definitions can be specified by compound workflow definitions and may optionally include a *data conversion function* which defines a mapping of data from the data format of the data source to that of the data sink.

Table 1 shows the parameter definitions of the workflow types `HICProcessing` and `EnterCase`, and of the block definition `checks`. In Table 2, the data flow definitions are listed and Fig. 5 graphically represents them using data flow arrows that connect graphical symbols representing complex workflow types and subworkflow declarations.

5.3 Behavioral Perspective

In the behavioral perspective, the focus is on when and under which conditions workflows have to be executed. Since we assume processing entities as being autonomous, activities (that are carried out by these processing entities) are regarded as black boxes whose internal structures are not visible. This means

Table 1. Sample parameter definitions

<i>Workflow Definition</i>	<i>In-Parameters</i>	<i>End States</i>	<i>Out-Parameters</i>
HICProcessing	-	accepted	-
		rejected	-
EnterCase	-	end	PolicyReference policy_ref, Cost cost, Treatment treatment, Diagnosis diagnosis
checks	PolicyReference policy_ref, Treatment treatment, Diagnosis diagnosis	ok	-
		not_ok	-

Table 2. Data flow definitions

<i>Workflow Definition Containing Data Flow Definitions</i>	<i>Source</i>	<i>Sink(s)</i>
HICProcessing	enter.end.policy_ref	checks.policy_ref, accept.policy_ref reject.policy_ref
	enter.end.cost	accept.cost, reject.cost
	enter.end.treatment	checks.treatment, accept.treatment, reject.treatment
	enter.end.diagnosis	checks.diagnosis, accept.diagnosis, reject.diagnosis
checks	checks.policy_ref	history.policy_ref
	checks.treatment	history.treatment, compatibility.treatment
	checks.diagnosis	compatibility.diagnosis

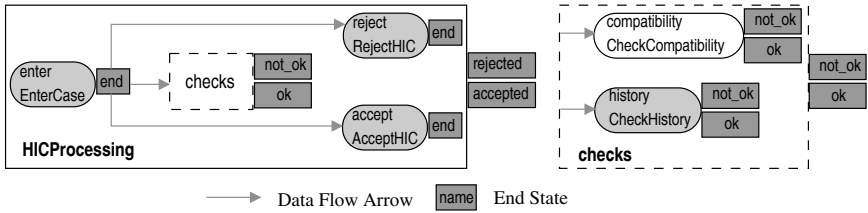


Fig. 5. Informational diagram: health insurance claim processing

that intermediate execution states of activities are not visible for the WFMS and that the WFMS can only control the start of activities.

In TRAMS, a workflow has one or more end states, where each end state represents a different outcome of the workflow. Each workflow runs through the execution states **Initiated**, **Executing** and finally ends in one of its end states.

When workflows have to be executed is defined by *guards*, which are logical expressions controlling the state transitions of workflows. Basic predicates of these expressions refer to properties of workflow instances. For a subworkflow declaration, a *start condition* can be defined, which is a guard on the transition leading from state **Initiated** to state **Executing**. A compound workflow definition can specify an *end condition* for each of its end states which are guards on the transitions leading from **Executing** to end states.

In start and end conditions, parameters of a workflow **wf** can be referenced using the following constructs:

- **wf.in_par** refers to the in-parameter named **in_par** of **wf**
- **wf.end_state.out_par** refers to the out-parameter **out_par** of **wf** that is associated to the end state **end_state** of **wf**

Furthermore, the following predicates can be used in start and end conditions:

- `state_name(wf)` evaluates to true if `wf` is in the state named `state_name`. Otherwise `state_name(wf)` evaluates to false.
- `start_time(wf)` is the point in time when `wf` has started
- `end_time(wf)` is the point in time `wf` has entered an end state

Note that certain attributes or parameters of a workflow may have unknown values during certain periods of time. For instance, the value of `start_time(wf)` is known only after `wf` has started, or the value of `wf.end_state.out_par` is only known after `wf` has entered the end state `end_state`. Therefore, conditions are three-valued logical expressions (similar to the ones used in FlowMark [16]) that can be evaluated to either `true`, `false`, or `unknown`.

Start and end conditions can be defined as follows. The start condition of a subworkflow declaration can refer to in-parameters, data variables, start time of its superworkflow, and attributes of sibling workflows (i.e., out-parameters, start and end times, and execution states). The end condition of a compound workflow `wf` can refer to in-parameters and the start time of `wf`, data variables defined by `wf`, and attributes of subworkflows (i.e., out-parameters, start and end times, and execution states).

In addition to the start and end conditions specifiable for a “normal” block definition, an iteration block definition includes a *repeat-until-condition* which determines at runtime how many times a new set of the subworkflows (called *iteration set*) defined by the iteration block has to be created and executed one after another. The data variables defined by an iteration block are created only once, while the iteration block is in the execution state `Initiated`. Thus, data flows among subworkflows belonging to different iteration sets can be defined using references to data variables.

A repeat-until-condition `ruc` can contain references to the same constructs as an end condition. A reference in `ruc` to an attribute or a parameter of a subworkflow contained in the iteration block for which `ruc` is defined always relates to the subworkflow contained in the iteration set that has been created last.

Besides the start and end conditions that are also specifiable by “normal” block definitions, a parallel block definition can define a *parallel-end-condition* for each of its end states. In order to illustrate the difference between an end condition and a parallel-end-condition, consider a parallel block `pb` with the end state `end_z`, and an end condition `ec` and a parallel-end-condition `pec` defined for `end_z`. In this case, `ec` specifies when a parallel set of `pb` may enter `end_z`, and `pec` defines when `pb` may enter `end_z`. The parallel-end-condition of a parallel block `pb` can involve existential and universal quantifiers, e.g.:

- `for_all x: success(x)` evaluates to true if all parallel sets of `pb` are in state `success`
- `exists x: failure(x)` evaluates to true if at least one parallel set of `pb` is in state `failure`

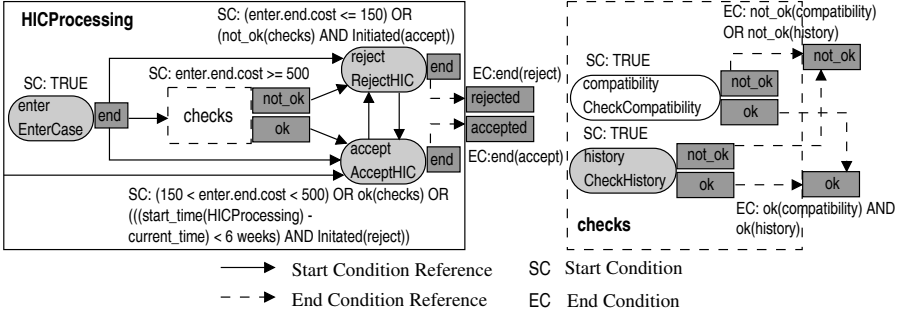


Fig. 6. Behavioral diagram: health insurance claim processing

Condition arrows allow to graphically represent the references to properties of workflow elements (here, a workflow element is either a complex workflow type, subworkflow declaration or data variable) included in start and end conditions. There are two kinds of condition arrows: start condition arrows and end condition arrows. A *start condition arrow* is a solid black arrow that points from a workflow element x to a subworkflow declaration y . Such an arrow means that the start condition of y involves one or more properties of x . Analogously, an *end condition arrow* is a dotted black arrow pointing from a workflow element x to the end state y of a compound WFD. The meaning of such an arrow is that the end condition associated to y involves one or more properties of x .

Figure 6 shows graphically the behavioral perspective of the HICProcessing workflow type.

5.4 Organizational Perspective

In the organizational perspective, the focus is on which processing entities carry out the activities of workflows. To that end, the organizational and the workflow schema have to be linked. Before we show how the assignment of activities to processing entities can be specified, we introduce two associations between processing entities and activity types. Processing entities provide the execution of activities of specific types (represented by the association *provides*). The association *executed_by* links an activity to the PE that actually carried out the activity.

The specification of which processing entities should carry out which activities is done by *filter functions* that are associated with activity variables. A filter function associated with an activity variable v maps the processing entities that provide the activity type of v to a subset of these processing entities, thus it filters out the desired processing entities. At execution time, the evaluation of a filter function results in a set of PE that either 1) contains exactly one PE, 2) is empty, or 3) contains more than one PE. In case 1), the respective activity is assigned to the only PE contained in the set. Case 2) has to be handled by

Table 3. Sample filter functions

<i>Activity Variable</i>	<i>Filter Function</i>	<i>Explanation</i>
enter	select r.processingEntity from r in PlaysRole where r.role.name = "Clerk"	filters out the processing entities that can play the role "Clerk"
compatibility	select p from p in ProcessingEntities where p.name="CompatibilityChecker"	filters out the processing entities named "CompatibilityChecker"

the enactment environment, e.g., by informing the workflow administrator. In case 3), several possibilities exist such as selecting randomly a PE from the set or choosing the PE with the currently lowest workload.

Filter functions are defined in OQL [5] and may involve references to the same constructs as a start condition (as defined in Sect. 5.3) and additional references to:

- attributes of organizational entities (e.g., “activity x has to be carried out by the processing entity named Peter”)
- relationships that exist among organizational entities (e.g., “activity x has to be carried out by a processing entity that belongs to the organizational unit sales_department”)
- previous assignments of activities to processing entities (e.g., “activity x must be executed by the processing entity that carried out activity z”).

Table 3 exemplarily shows the filter functions for two activity variables of the workflow type HICProcessing. The filter functions for the other activity variables can be defined in a similar way.

6 Related Work

In recent past, a plethora of workflow modeling approaches has been proposed. In this section related work is considered as far as relevant for this paper.

Most existing approaches either do not support organizational modeling at all or use a fixed organizational schema. In contrast to our approach, they do not support the specification of user-defined organizational entities and relationships. Exceptions are the approaches described in [3] and [13]. In [3], organizational modeling and the organizational perspective of workflows are considered, but not as part of a complete workflow specification approach. Furthermore, only binary organizational relationship types can be defined as opposed to our approach, where n-ary relationship types can be defined. We regard the MOBILE workflow model as described in [13] not as a concrete specification approach, but as a framework from which concrete specification approaches could be developed.

Similar to the TRAMs-approach, others also rely on the notion of execution state e.g., [15,16,18,19]. These approaches differ with respect to the number

of definable states and the way in which transitions are specified. One group (e.g., FlowMark [16] and [4]) assumes a fixed set of states and therefore does not allow to represent different workflow results by different states. The second group allows the definition of an arbitrary number of states (e.g., Mentor [18], and METEOR [15]). This leads to a state-centered view of workflows that is counter-intuitive since most reasoning about business processes and workflows is usually activity-centered. Furthermore, some approaches use imperative constructs (such as ECA-rules in Mentor) for controlling state transitions. In TRAMs, the combination of fixed state transitions and multiple end states supports an activity-centered view capable of explicitly expressing different workflow results. The specification of permissible state transitions is declarative because logical expressions are used for that matter.

States and end states are thus the means to specify control flow. Petri-net based approaches (e.g. Information Control Networks [9], INCOME/WF [17], FUNSOFT Nets [7,8]) do not support states explicitly. They model control flow through the token flow, and the execution state of a workflow is represented by the distribution of the tokens in the corresponding Petri-net. In particular, these approaches mix different perspectives (informational, structural, behavioral) as these are expressed using the same constructs.

7 Conclusion

In this paper, we have described the TRAMs approach towards workflow modeling. This approach allows to specify the organizational structures and relationships as well as the workflows to be executed. Workflow specification supports all the relevant perspectives.

The TRAMs workflow model is modular in the sense that the various elements (workflows, organizational entities, etc.) can be specified independently from each other, and in that no assumptions are made in which context the elements are going to be used (for instance, activity assignment, control and data flow are not specified with the activity itself, but only when the activity is used within a workflow). TRAMs is activity-centric in that it allows to “think” in terms of activities/workflows and their results instead of states or transitions. Furthermore, abstraction is supported, e.g., by specifying conditions as logical expressions instead of as implementation-level concepts such as ECA-rules. Therefore, summarizing, the contribution of the TRAMs-approach towards specification is that it — unlike other approaches — better supports a component-oriented way of developing workflow systems.

A graphical editor for workflow specifications is operational. Specifications developed using this editor can be mapped onto the workflow engine EVE [11], i.e., specifications can be made executable by EVE. In our future work, we will extend the specification editor towards a full-fledged development environment. In particular, a repository system [1] is under development that maintains a repository of reusable workflow specifications. In addition to the graphical editors, tools will be developed that support the selection, adaption, and integration

of reusable specifications for new workflow types. Ultimately, we will address the specification of flexible recovery mechanisms for TRAMs in our future work.

References

1. P.A. Bernstein and U. Dayal. An Overview of Repository Technology. *Proc. 20th Int'l Conf. on Very Large Data Bases*, Santiago, Chile, September 1994. 276
2. B.W. Boehm. A Spiral Model of Software Development and Enhancement. *ACM Software Engineering Notes*, 11:4, 1986. 264
3. C.J. Bussler. Policy Resolution in Workflow Management Systems. *Digital Technical Journal*, 6:4, Fall 1994. 264, 275, 275
4. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual Modeling of Workflows. In *Proc. 14th Int'l Conf. on Object-Oriented and Entity-Relationship Modelling (OO-ER'95)*, Gold Coast, Australia, December 1995. 276
5. R. G. G. Cattell and D.K. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, 1997. 267, 275
6. B. Curtis, M.I. Kellner, and J. Over. Process Modelling. *Communications of the ACM*, 35:9, September 1992. 268
7. W. Deiters and V. Gruhn. The FUNSOFT Net Approach to Software Process Management. *Int'l Journal on Software Engineering and Knowledge Engineering*, 4:2, 1994. 276
8. G. Dinkhoff, V. Gruhn, A. Saalman, and M. Zielonka. Business Process Modeling in the Workflow Management Environment Leu. *Proc. 13th Int'l Conf. on the Entity-Relationship Approach (ER '94)*, Manchester, UK, December 1994. 276
9. C. A. Ellis and G. J. Nutt. Modeling and Enactment of Workflow Systems. *Proc. 14th Int'l Conf. on Application and Theory of Petri Nets*, Chicago Illinois, USA, June 1993. 276
10. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:2, April 1995. 263
11. A. Geppert and D. Tombros. Event-based Distributed Workflow Execution with EVE. *Proc. Middleware '98*, The Lake District, England, September 1998. 276
12. C. Hagen and G. Alonso: Flexible Exception Handling in the OPERA Process Support System. *Proc. 18th Int'l Conf. on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998. 265
13. S. Jablonski and C. Bussler. *Workflow Management. Modeling Concepts, Architecture, and Implementation*. International Thompson Computer Press, London 1996. 263, 275, 275
14. M. Kradolfer and A. Geppert. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. *Proc. 4th Int'l Conf. on Cooperative Information Systems (CoopIS '99)*, Edinburgh, Scotland, September 1999. 265
15. N. Krishnakumar and A. Sheth. Managing Heterogeneous Multi-System Task to Support Enterprise-Wide Operations. *Distributed and Parallel Databases*, 3:2, 1995. 275, 276
16. F. Leymann and W. Altenhuber. Managing Business Processes as an Information Resource. *IBM Systems Journal*, 33:2, 1994. 273, 275, 276
17. A. Oberweis, R. Schaetzle, W. Stucky, W. Weitz, and G. Zimmermann. IN-COME/WF — A Petri Net Based Approach to Workflow Management. *Proc. Wirtschaftsinformatik '97*, Berlin, February 1997. 276

18. D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz-Dittrich. The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. *Proc. 12th Int'l Conf. on Data Engineering*, New Orleans, February/March 1996. 275, 276
19. Workflow Management Coalition. Terminology & Glossary, Issue 3.0. Technical Report WfMC-TC-1011, February 1999. 275

From CASE to CARE (Computer-Aided Requirements Engineering)*

Oscar Pastor, José H. Canós, Isidro Ramos

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Camí de Vera, S/N, E-46071 València (Spain)

Tel: +34 96 3877350; Fax: + 34 96 3877357

e-mail: { opastor | jhcanos | iramos }@dsic.upv.es

Abstract

Software Engineering is still lacking methods which are capable of properly capturing the relevant System properties in the problem space and at the same time generating a correct representation (a final software product) in the solution space. This product must be functionally equivalent to the system specification and obtained in an automated way. The design of methods of this kind and the development of associated support tools is one of the current challenges in the field.

This leads us to real CARE Computer-Aided Requirements Engineering environments, where the replacement of the S (of CASE) by the R is very meaningful: we want to focus on the Requirements Engineering process to capture properly the relevant system properties. Our CARE advanced tools will convert these “patterns” of behavior into the most convenient representation in a given software development environment, by properly “compiling” the system specification following a set of mappings between conceptual patterns and software representations.

To do it, a formal basis and a model with the necessary expressiveness are needed in order to do this. The contribution of this paper is the object-oriented OASIS model as the basic model for such a CARE environment, together with a method for software production –the OO-Method- as an operational implementation of these ideas.

1. Introduction

Conventional CASE tools have traditionally dealt with the software production process by using different abstractions and notations for “*what*” the system is (problem space) and “*how*” it is to be represented in a particular software development environment (solution space). This fuzzy strategy has always originated doubts about the real utility of these CASE environments which are intended to ease the software production process. It has often been experienced that the use of CASE tools, rather than easing the solution of a problem, creates a new problem besides the original one: how to properly use the different notations provided by the tool in a set of diagrams, which often have overlapping semantics.

* This work has been supported by the CICYT under MENHIR/ESPILL TIC 97-0593-C05-01 Project and by DGEUI under IRGAS GV97-TI-05-34 Project.

Structured Analysis techniques use two different abstractions to deal with statics and dynamics. On one hand, an Entity-Relationship-like data model is provided to represent the static system architecture including data components, their definition and their relationships. On the other hand, a process model is necessary to properly specify the system behavior. Object Oriented (OO) technology has emerged as an important advance by introducing the notion of object as the only abstraction, (including both statics and dynamics) to represent reality. However, the corresponding OO CASE tools have not been able to clarify the software production process.

Nowadays, we find a set of what could be called *conventional* OO methodologies, coming from practical use in industrial software production environments, which do not have a formal basis and which often use classical structured concepts together with the introduction of OO features ([1,2,3,4,5]). Recent proposals are trying to create a unified framework for dealing with all the existing methods (e.g. UML [6]). They have the implicit danger of providing users with an excessive set of methods and notations whose semantics overlap.

On the other hand, in their quest for a solution, other authors have proposed the use of OO *formal specification languages* (Oblog [7,8], Troll [9,10], Albert [11], OASIS [12, 13]). These languages have a solid mathematical background and deducible formal properties such as soundness and completeness.

Neither of these approaches offers an acceptable individual solution. Formal methods are hard to use and do not reach industry. Conventional methods are so ambiguous that in practice they often generate the same “hostile” reaction for different reasons. However, if both approaches have some good properties when dealing with the problem of elaborating quality software, a simple question arises: why not blend their good properties leaving the useless ones out?

Software Engineering is still lacking a method which is capable of:

1. properly capturing the relevant system properties in the problem space
2. generating a solution in the form of an end software product in the solution space, which is functionally equivalent to the system specification and is obtained in an automated way.

This leads us to a Computer-Aided Requirements Engineering (CARE) environment, where the replacement of the S by the R is very meaningful: we want to focus on the Requirements Engineering process to capture properly the relevant system properties. Our advanced CARE tools will convert these patterns of behavior into the most convenient representation in a given software development environment, by properly “compiling” the system specification.

This approach creates new avenues for dealing with traditional Software Engineering problems that have not yet found a clear solution in the conventional CASE world, such as reuse, schema evolution or automated code generation. If the specification expressiveness is based on a notation which is close to the problem space, the reusability and evolution of such a specification will be easier. If we determine the software components that correspond to a finite set which includes the potentially valid behavioral patterns, code generators could be implemented by programming the mappings that convert the quoted behavioral patterns in their corresponding concrete software representations.

Our research work has been directed towards designing and implementing an OO software production environment whose objective is to combine the pragmatic

aspects attached to the so-called conventional methods with the good formal properties of the OO specification languages. To do it, a graphic, OO conceptual modeling environment that collects the relevant system properties for building a formal, textual OO specification in an automated way, is introduced.

In contrast to other works in this area ([14,15]), we focus on the definition of a concise execution model and on the mapping between the specification language and the execution model notions. This makes possible to build an operational implementation of a software production environment allowing for real automated prototyping, by generating a complete system prototype (including statics and dynamics) in the target software development environment. A CARE workbench which supports this working environment in a unified way is currently available for prototyping purposes. In this paper, we will show that with the formal OASIS model and the corresponding operational implementation of the OO-Method, we support previous works based on the idea that we can significantly reduce the complexity of advanced-application specification and implementation by using a model-equivalent language (a language with a one-to-one correspondence to an underlying, executable model [16]). This model-equivalence is obtained in our case by defining a precise set of conceptual modeling patterns, and the mapping that converts them into concrete software representations.

To accomplish these aims, the structure of this paper is the following. After the introduction, we describe the most relevant features of the OASIS object oriented model, introducing the basic ideas on OO conceptual modeling that are the basis of the work. Next, we introduce the main contributions of the OASIS model to give precise solutions to traditional Software Engineering problems such as schema evolution or methodological guidance from problem space notions to solution space representations. An operational implementation based on the OO-Method – the methodological approach attached to OASIS – is introduced in section 4. We will focus on how to represent a conceptual model in a particular software development environment according to an abstract execution model, which will fix the operational steps to follow when we want to give a concrete system implementation. A software prototype which is functionally equivalent to a system specification can be obtained in the context of the methodology. We describe the code generation strategy used. Conclusions and the list of references end the paper.

2. The OASIS Model

Prior to the presentation of the OASIS model, let us briefly introduce Dynamic Logic, which will be used in this paper to describe it.

2.1 Formal Framework: Dynamic Logic

Dynamic Logic (DL) is a modal logic proposed by David Harel [17] in the late 70's as a formalism for reasoning about those programs that check and modify a given environment. The main goal was to describe in a formal way the effect of programs execution as a first step for further formal reasoning about them. Although Harel's work on DL is very extensive, we present below the dynamic formulae that are relevant to our purposes.

A Dynamic formula is expressed as $\phi [p] \psi$ where ϕ and ψ are first order formulae, p represents a program (in a general sense) and $[_]$ is the necessity operator.

The intuitive meaning of such a formula can be stated as follows: “if ϕ holds, then after execution of the program p , ψ must hold”.

Dynamic formulae (as opposed to first order formulae) have a special singularity. The latter are interpreted in a single interpretative structure, whereas the former need several structures (in particular, 1st-order logic is a subset of DL). This important difference can be understood by taking into account that dynamic formulae are used to describe an evolutionary process, that is, a certain change of state. Thus, a formal dynamic theory is interpreted by using a Kripke structure $K=(\Omega, \omega_0, \rho)$ where:

- Ω is a set of possible worlds, each one of which is defined as a first order interpretative structure,
- ω_0 is the initial world,
- ρ is an accesibility relation among worlds (see Figure 1).

The effect of the execution of any program can be described by using this language. For our purposes, we use DL to describe the evolution of objects. In other words, we use it to describe the state changes in objects that take place as a reaction to certain external stimuli (namely events¹) that will play the role of programs in classical DL. We present the OO model in which our work is based, first from an intuitive point of view and later through more precise concepts.

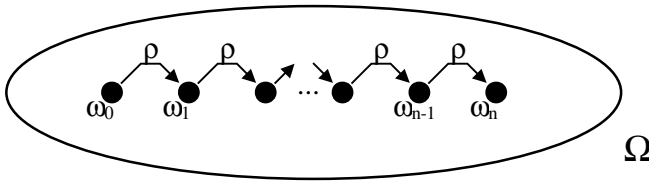


Figure 1 Simple Kripke structure

2.2 Basic Concepts

In the OASIS model, Information Systems (IS) are considered as societies of communicating objects that evolve through state changes following certain behavior rules. An *object* is observed in order to know some of its structural properties, which are represented in the model by the notion of *attribute*. Each object is referred to by means of its *identifier*, which distinguishes it from any other object in the IS. The *state* of an object is given at any moment by the set of values of its attributes (represented as a 1st-order presentation). Thus, a change in any of the attribute values will lead to a change in the object's state. The value of an attribute may change due to the occurrence of an *event* in the life of the object; the way the value of an attribute changes after an event occurrence is described using valuation rules. In addition, a set of integrity constraints restrict the possible states and state sequences or *traces* that an object can reach during its life. Object lives are maximal traces.

¹ similar notions in other models are messages, events, transactions, etc.

A formal definition of the OASIS model can be found in [13]. However, the following concepts are relevant to the aim of this paper.

Object Types. An object type is defined as a tuple $T=(A,X,F,P)$, where:

- A is a set of *attributes*, which may be constant, variable and derived; for a given object, constant attributes take their values when the object is created and do not change during its life, whereas the values of variable attributes change with event occurrences; finally, derived attributes take their values from other attributes according to some derivation rules.
- $X = X_s \cup X_a$ is a set of *events*; X_s are the *services* provided by the instances of the type, and X_a are the *actions* that might be requested to other objects in the system, even to themselves. For every action there must be a service with the same name in some type of the system.
- $F = F_v \cup F_d \cup F_p \cup F_t$ is a set of *formulae* which capture different aspects of the OASIS behavioral model; F_v are the *valuation* formulae that state how attribute values are changed by events; F_d is a set of *derivation* rules associated to derived attributes; F_p are the event *preconditions*, that define the valid states of an object for an event occurrence to be relevant (i.e., to actually change the object's state); F_t are the *triggering rules*, that define part of the reactive behavior of the OASIS objects by stating events that must be activated when a condition is satisfied; finally, F_c are *integrity constraints* which are expressed as temporal formulae that must hold over any sequence of states of an object.
- P is a term built over the Basic Process Algebra (BPA) defined in [18]; every actual life of an object must be a ground process term of P .

Objects and Classes. An *object template* is a triple (T, I, P) , with T being a type, I an identification function (a set of pairs (identifier, instance)) and P the public part of T that for a type states which attributes and/or events² are accessible to the objects in the system. So, an object o in a state is a triple (Σ, i, S) , Σ being a template, $i = I(o)$ the identifier of o obtained by applying the identification function $I \in \Sigma$ to it, and S the state of the object. S is described using first-order formulas (precisely, a set of ground atoms, one per attribute in the type of the object).

A *class* adds to the object template the aspects of *factory* (i.e. creation and deletion of objects) and *extent* that also include some other object models [19]. Classes are represented as objects whose template includes the properties listed below:

- object factory properties are those that allow a class object to create new objects called instances of the class. Of these, we describe the following:
 - a collection of attributes whose current values define the template to be used to create instances; examples are *constant-attribute-list*, *derived-attribute-list*, *private-event-list*, and *constraint-list* ([20]).
 - two special events, *new* and *destroy*, used to create (using the template defined by the factory attributes) and destroy instances, respectively

² in the sequel, we will refer to both attributes and events as properties

- an attribute *next-oid* whose value is used by *new* to assign an identifier to the created object, and which is updated after its use
- collection properties: an attribute *extent* which at any moment holds the set of identifiers of the alive instances of the class, i.e. objects created by the class object that have not yet been destroyed.

Hence, the schema meta-information is included in the model as part of the state of the OASIS classes seen as objects. This is a very important property that will be used later in this paper to include schema evolution as a built-in property of the OASIS model (see section 3).

Class Hierarchies. The OASIS classes can be elementary or complex; complex classes represent either generalization/specialization or aggregation relationships. A description of these relationships is outside the scope of this paper, due to the fact that the abstract execution model we introduce below is orthogonal to this dimension. We refer the reader to [13] for a detailed description of the OASIS class relationships.

3. The OASIS Approach to the Classical Software Engineering Problems.

The OASIS model allows the specification of the structure and behavior of an Information System in great detail. The structural aspects are captured as in other OO models. However from the behavioral point of view the OASIS expressiveness is greater. The OASIS model is more complex in the sense that not only the signature of the events is given in each object type, but also the way they are allowed to happen. That is, the patterns of allowed behaviors of the objects are explicitly captured in the model following two approaches:

- *State dependent specification*: event preconditions and triggering relationships capture the deontic concepts of permission and obligation, respectively.
- *State independent specification*: the allowed patterns of behavior are given by using terms of the Basic Process Algebra.

This richer expressiveness allows us to capture more semantics of the problem domain; the price for this is to get a more complex model to understand and use, although the benefits we obtain compensate for this problem, specially if we combine the process specification capabilities with the metalevel aspects as they are introduced in the OASIS model (see below).

To be homogeneous and consistent, the OASIS model views all class objects as instances of another class, namely the OASIS *Metaclass*. The OASIS *Metaclass* is the only predefined object in the model and is an instance of itself (closing in this way the circularity induced by the model). All other objects (either class objects or plain instances) can be created dynamically through the event *new* associated to any class object. This has implications in the quality of the software artifacts obtained by using the OASIS Model. We illustrate them at two levels, namely the *object* or Information System (IS) level and the *metalevel*.

3.1 The Object Level

At an object level, the conceptual models we get are semantically richer and more complete because the behavior of an IS is included in the conceptual model. As OASIS has strong logical and categorical foundations, the operational semantics of DL makes OASIS specifications executable. Therefore, it is no longer necessary to use a host language (like C++, Smalltalk or others) that use a different epistemological background, a different modeling metaphor and different software tools (Compilers or Interpreters) to capture this behavior.

From a Software Engineering point of view, we obtain homogeneity in the sense that a unique language which technically implements the model expressiveness acts as the Object Definition Language (ODL), as the Object Query Language (OQL) and finally, as the Object Manipulation Language (OML) [20]. The ODL is a class definition language. The OQL is the technical implementation of the declarative language of the state logic used [21] and the OML is the language supporting the process expressiveness of the model. An OASIS object (which is a process) can be seen as the program (in the classical sense) that the quoted object represents. This is usual in some OO+Logic Programming Approaches (e.g. [22]). In this way, a program in OASIS is an active object implementing the obligation of fulfilling its process part.

The homogeneity in the conceptual models produced by using OASIS allows for the definition of a unique translation schema for any target environment: the Execution Model (EM) (see 4.1 below). Its introduction allows for a true implementation of the Automatic Programming Paradigm, fulfilling the gap between the problem space and the solution space descriptions.

The software tools implementing these ideas are clearly an advancement over the classical CASE approach. A companion methodology, called OO-METHOD [23], is added to OASIS and is introduced in section 4. As will be shown, advanced visual programming environments are used in OO-METHOD to produce the Conceptual Models that will automatically be mapped to implementations in OO programming languages.

3.2 The Metalevel

As in many other approaches, the classes composing a given Conceptual Model are instances of a metaclass (the OASIS Metaclass or OM for short). What is new in the OASIS Model is the dual view of these metaobjects³:

- As classes, they have a name, a template (T1) which includes factory services and the extent.
- As (meta)objects, they have an oid and a template T2 whose attribute set defines the properties of their instances, and a dynamic behavior given by a signature of events and a process; these events change T1 in accordance with the formulae in T2. Note that T2 defines a schema evolution mechanism.

In general, T1 and T2 are different for each class. The only exception is OM, for which T1=T2. This way we cut the infinite circularity of other metaclass-based

³ In the sequel, we will use the terms class and metaobject interchangeably to refer to the instances of the metaclass OM

approaches. It is interesting to note that the process part of OM captures the method to be followed by a system designer to build an IS. Another interesting feature comes from the fact that the extent of OM is the repository of applications.

The coding of a model in itself is a form of reflectivity that provides us with a simple and elegant way to cope with the classical Software Engineering problems of software evolution, software reuse, software versioning, software process formalization, true methodological aids and guidance, etc. Although the unique model used in our case is OASIS, other models could be used simultaneously if several metaclass were defined and included in a software production environment (a modern CARE tool). These models can be independent or related by the relationships provided by the model (e.g. is-a, part-of, etc.).

3.3 The Problems and the Solutions

Several Software Engineering problems are presented and the solutions within the OASIS Model are outlined in the following points. A set of these solutions has already been implemented. Others have been clearly stated and specified within the model but not implemented and others have only been enunciated. The aim of this section is to illustrate the good properties of the OASIS model within the Software Engineering field and how future CARE environments will be produced in the near future.

Software Evolution. An application (a conceptual schema in OASIS) is defined as the state of a given class that can be changed at any moment by the user through the services offered by the metaobject. The changes resulting from the occurrence of the event in the metaobject state are described in a declarative way in the OM specification. On the one hand, some changes make template T2 evolve, and on the other hand, other changes must affect to extent in order to remain compliant with T2.

Two implementations of the evolution services have been performed. One of them [25] uses Transaction Frame Logic [24] as the underlying logic language and the other one has been implemented on top of a Clausal Dynamic Logic programming environment.

Software Versioning. Software evolution is motivated by the dynamic of business rules in the problem space. If we try to keep track of the history of the different Conceptual Models (each of which is valid within a period of time) we need a time stamping of the different states that the class defining the conceptual schema reaches during its life. For this purpose, a time model based on Kowalski's Event Calculus is used in OASIS so that each state is stamped by the periods of time in which the conceptual model it represents is valid. Then, one may retrieve past versions of schemas by means of temporal queries [20].

Obviously, the same time model can be used at the object level to implement classical temporal data models and the historic files. *Method Aids and Guidance.* Current industrial CASE tools and Companion Methodologies provide a set of visual languages to describe the Conceptual Models and a sometimes tedious set of heuristics that the developer has to follow to be compliant with the method rules. This is one of the reasons for the traditional CASE underachievement. In practice, the developer develops software which is only partially compliant with the companion method.

It is clear that the only way in which a software development method can be successfully applied is by implementing it in the computer by means of a set of aids

and guidance tools. Within OASIS, the process part of the metaclass OM clearly and unambiguously fixes the way in which software must be built. The implementation of these ideas within a CARE Environment is the goal of the MENHIR Project that unites five spanish university teams and four software companies for 3 years (1998-2000).

A partial implementation of these ideas is the OO-METHOD CASE tool whose basic concepts are presented in the following point. We will emphasize how the set of services provided by the metaobject OASIS fixes a particular implementation where all these services are provided in a conventional CARE environment and where the Conceptual Model is converted into a final software product as a value-added feature, by implementing an abstract execution model that guides the process of implementing OO-Method Conceptual Models.

4. The Methodology

The *OO-Method* is an Object-Oriented Software Production Methodology whose phases are shown in Figure 2. Basically, we can distinguish two components: the conceptual model and the execution model. When facing the conceptual modeling step of a given Information System, we have to determine the components of the object society without any implementation concern. The problem at this level is to obtain a precise system definition (the conceptual model). Once we have an appropriate system description, a well-defined execution model will fix the characteristics of the final software product in terms of user interface, access control, service activation, etc.; in short, all the implementation-dependent properties.

In this context, we start with an Analysis step where three different models are generated: the Object Model, the Dynamic Model and the Functional Model. They describe the Object Society from three complementary points of view within a well-defined OO framework. For these models, we have preserved the names used in many other well-known and widely-used OO methodologies, even if the similarities are purely syntactic as can be seen throughout this paper.

The OO-Method implements the set of services provided by the metaclass OM in a conventional way to build a conceptual model. For every service there is an option implemented in the most convenient way in the tool: for instance, for the *create attribute* service, a friendly and convenient implementation is provided in the context of a class specification.

From these analysis models, a corresponding formal and OO OASIS specification (the OO-Method design tool) can be obtained in an automated way. This is done through an automatic translation process. The resultant OASIS specification acts as a complete system repository, where all the relevant properties of the component classes are included. This is an important contribution of the OO-Method approach: the analysis models are intended to capture only the relevant system information, and what is considered relevant is fixed by the semantics of the OASIS specification language.

According to the execution model, a prototype which is functionally equivalent to the specification is built in an automated way. This may be done in both declarative (Prolog-based) and imperative environments (specially those visual OO programming environments that are widely used nowadays). The code generation strategy is independent of any concrete target development environment, even if

currently our selected environment for automated code generation are Visual C++, Delphi, Java, Visual Basic and PowerBuilder.

The basic characteristics of the three models (object, dynamic and functional) that constitute the conceptual model are explained in [26]. Now, we introduce the execution model features to explain how the conversion strategy from the former to the latter is done within OO-Method.

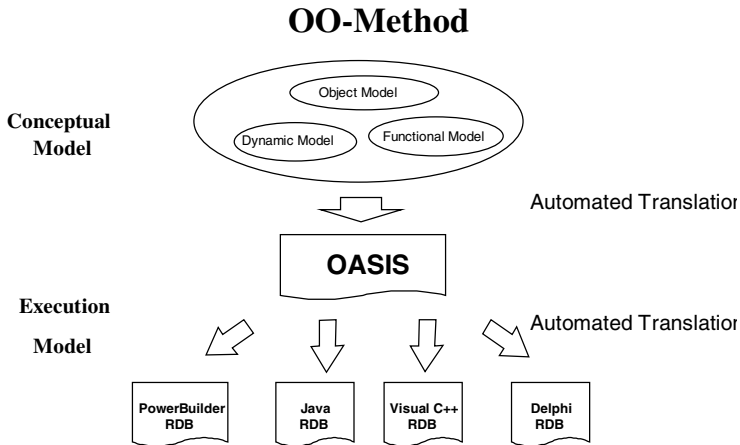


Figure 2. Phases of OO-Method.

4.1 Execution Model

Once all the relevant system information in the specification that we have called conceptual model is collected, the execution model has to accurately state the implementation-dependent features associated to the selected machine representation of the object society. More precisely, we have to explain the pattern to be used to implement object properties in any target software development environment.

Our idea at this point is to give an abstract view of an execution model that will set the programming pattern to be followed when dealing with the problem of implementing the conceptual model. This execution model has three main steps:

1. *access control*: first, the user logging in the system has to be identified as a member of the corresponding object society.
2. *object system view*: once the user is connected, he must have a clear representation of which classes he can access. In other words, his object society view must be stated, precisely defining the set of object attributes and services he will be allowed to see or activate, respectively.
3. *service activation*: finally, after being connected and having a clear object system view, the user will be able to activate any available service in the user's world view. Among these services, we will have event or transaction activation served by other objects, or system observations (object queries).

Any service execution is characterized by the following sequence of actions:

1. *object identification*: as a first step, the object acting as server has to be identified. This object existence is an implicit condition for executing any service, except if we are dealing with a *new*⁴ event. At that moment, their values (those that characterize its current state) are retrieved.
2. *introduction of event arguments*: the rest of the arguments of the event being activated must be introduced.
3. *state transition correctness*: we have to verify in the State Transition Diagram associated to every class that a valid state transition exists for the selected service in the current object state.
4. *precondition satisfaction*: the precondition associated to the service that is going to be executed must hold. If not, an exception will arise, informing that the service cannot be activated because its precondition has been violated.
5. *valuation fulfillment*: once the precondition has been verified, the induced event modifications are effective in the current object state.
6. *integrity constraint checking in the new state*: to assure that the service activation leads the object to a valid state, we must check that the (static and dynamic) integrity constraints hold in this final resulting state.
7. *trigger relationships test*: after a valid change of state, and as a final action, the set of rules condition-action that represents the internal system activity have to be verified. If any of them holds, the corresponding service activation will be triggered. It is the analyst's responsibility to assure the termination and confluence of such triggers.

The previous steps guide the implementation of any program to guarantee the functional equivalence among the object system description collected in the conceptual model and its reification in a software programming environment according to the execution model.

Next, we are going to present the code generation strategy used in the implementation of the previous execution model in a graphic environment, which opens up the possibility of creating a CARE tool that, starting from a set of graphical OO models obtained during the conceptual modeling step (according to OO-Method) can generate a functional software prototype at any time.

4.2 Code Generation Strategy

Once an abstract execution model has been defined, we will have different concrete implementations of this execution model for different software development environments. It is important to remark that the representation of the conceptual model in the selected execution model is done according to the principles introduced above, thus generating a prototype in an automated way by adapting the code generation strategy that we present to the particularities of the corresponding target development environment.

⁴ Formally, a new event is a service of a metaobject representing the class, which acts as object factory for creating individual class instances (see section 3)

Using a conventional graphical system, the code generation process creates four types of windows as can be seen in Figure 3:

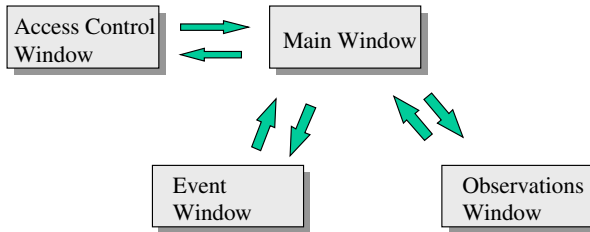


Figure 3. Overview of the generated code structure.

- *Access Control Window*: this is the log-in window (Figure 4), where the corresponding active user has to be identified. This is done by introducing its object identifier, class name and password. The identification is verified on the database to ensure that the object exists. Once the object is incorporated to the system, it will see the available system class services through menu items of the main menu.

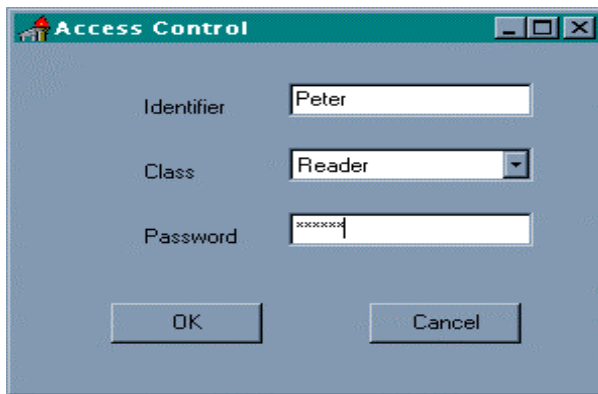


Figure 4. Access Control Window

- *Main Window*: it characterizes the system view that the connected object has (Figure 5). All the services of the classes are requested through it. For instance, in a simple library system, we will have as main options in the menu bar *loans*, *books*, *readers* and so on, representing the system classes we can access. Generally speaking, it has the following options:
 - the typical File item option of conventional graphical applications.

- for every class, a pull-down menu including an item for observations (queries), a section with its subclasses (if any) and a last section with the available class services. For instance, for the book class, we can activate the observations option, or the *new-book*, *loan*, *return...* events
- an interactions item, which allows for the activation of global interactions.

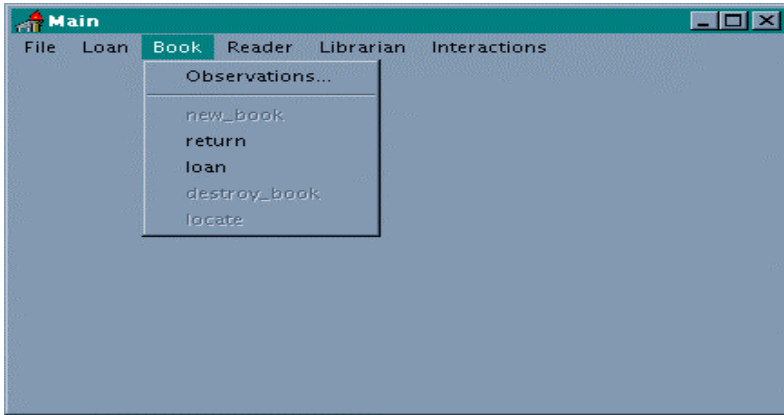


Figure 5 Main Window.

- *Event window*, where the corresponding arguments are introduced and the induced actions are executed through the OK control button. It has a script associated to it including the event pre/post conditions, specified in the OO-Method Conceptual Model and which are properly represented in the target software development environment.
- *Observations window*: this screen is intended to be a Query By Example pattern where the user can see the results of any query made about the current object state.

This is a precise strategy for executing OO-Method conceptual models, that will guide any concrete implementation where the particular aspects of the corresponding target development environment should be taken into account.

5. Conclusions

The provision of automated software engineering tools to support the whole software development process is very important in the promotion and adoption of the process and its associated methods. The development of these tools is a significant software engineering task. In this paper, such an environment is introduced using what we call CARE technology, based on a formal object-oriented model (OASIS) and a method (OO-Method) for capturing system requirements and converting them into executable software representations in a target software development environment.

Traditional approaches to the problem of producing software of quality are characterized by uncertainty, complexity and excessive effort to finally produce doubtful results: deciding whether a conceptual pattern with appropriate expressiveness exists, choosing between several possible representations apparently meant for the same purpose, obtaining a correct software representation which is functionally equivalent to the conceptual model, etc. are common problems solved in OO-Method by providing a precise set of conceptual modeling patterns (that have a formal counterpart in the OASIS model), their graphical representation and their corresponding software representation. Research work is being developed to

- (a) introduce interface patterns in the specification of the OO-Method Conceptual Models,
- (b) to improve the quality of the final software product obtained in an automated way and
- (c) to implement more accurately the ideas of the Oasis Metaclass presented in this paper to have more advanced CARE environments dealing properly with schema evolution and method guidance.

References

- [1] Wirfs-Brock R., Wilkerson B., Wiener L. *Designing Object Oriented Software*. Englewood Cliffs, Nj. Prentice-Hall. 1990
- [2] Rumbaugh J. et al. W. *Object Oriented Modeling and Design*. Englewood Cliffs, Nj. Prentice-Hall. 1991
- [3] Jacobson I. et al. G. *OO Software Engineering , a Use Case Driven Approach*. Reading, Massachusetts. Addison –Wesley
- [4] Booch, G. *OO Analysis and Design with Applications*. Addison-Wesley, 1994
- [5] Coleman D., Arnold P., Bodoff S., Dollin S. Gilchrist H., Hayes F., Jeremes P. *Object-Oriented Development; The Fusion Method*. Prentice-Hall 1994.
- [6] Booch G., Rumbaugh J., Jacobson I. *UML. v1*. Rational Software Co., 1997.
- [7] Sernadas A., Sernadas C., Ehrich H.D. *OO Specification of Databases: An Algebraic Approach*. In P.M. Stocker, W. Kent eds., proc of VLDB87, pages: 107-116, Morgan Kauffmann, 1987
- [8] ESDI S.A., Lisboa. *OBLOG CASE V1.0- User's Guide*
- [9] Jungclaus R., Saake G., Sernadas C. *Formal specification of Object Systems*. Eds. S. Abramsky and T. Mibaum Proceeding of the TapSoft's 91, Brighton. Lncs. 494, Springer Verlag 1991, pgs: 60-82.
- [10] Hartmann T., Saake G., Jungclaus R., Hartel P., Kusch J. *Revised Version of the Modeling Language Troll (Troll version 2.0)*. Technische Universitat Braunschweig, Informatik-Berichte, April 1994
- [11] Dubois E., Petit M., Wu S. *ALBERT: A Formal Agent-Oriented Requirements Language for distributed Composite Systems*. In Proc. CAiSE'94 Workshop on Formal Methods for Information Systems Dynamics, pages: 25-39, University of Twente, Technical Report, 1994
- [12] Pastor O., Hayes F., Bear S. *OASIS: An object-oriented specification language*. In P. Loucopoulos, editor, Proceedings of the CAiSE'92 conference, pp. 348-363, Berlin, Springer, LNCS 593 (1992).

- [13] Pastor O., Ramos I. *Oasis 2.1.1: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*, October 95 (3rd edition).
- [14] Wieringa, R.J., Jungclauss, R., Hartel, P., Hartmann, T., Saake, G., OMTROLL Object Modeling in TROLL. Proceedings of the International Workshop on Information Systems - Correctness and Reusability (IS-CORE'93). Hannover, September 1993. Udo W. Lipeck, G.Koschorrek (eds.).
- [15] Jackson R.B., Embley D.W., Woodfield S.N. *Automated Support for the Development of Formal Object-Oriented Requirements Specification*. In Proceedings of CAiSE-94 Conference. Utrecht, The Netherlands. Lecture Notes in Computer Science; v. 811; p.135-148. 1994.
- [16] Little S.W., Embley D.W., Woodfield S.N. *Unifying Modeling and Programming Through an Active, Object-Oriented, Model-Equivalent Programming Language*. In Proceedings of 14th International Conference on Object-Oriented and Entity-Relationship Modeling (OOER'95). 13-15 Dec 1995. Gold Coast, Australia. Lecture Notes in Computer Science v. 1021; p. 55-64. 1995.
- [17] Harel, D., *First-Order Dynamic Logic*, Springer-Verlag, 1979.
- [18] Wieringa R.J. {em A Formalization of Objects using Equational Dynamic Logic}, 2nd Conference on Deductive and OO Databases, pages 431-452, Springer 1991, Lecture Notes in Computer Science 566.
- [19] Catell, R. G. G. (ed.), *The Object Database Standard: ODMG-93, Release 1.1*, Morgan Kaufmann Publishers, 1994.
- [20] Canós,J.H.;Ramos, I.;Lozano,M.D. *KAOS: an Object-oriented software tool for the objects definition, updating, querying and programming in an object-oriented environment*. Proc of the IEEE 1998 Canadian Conference on Electrical and Computer Engineering, IEEE Press, 1998
- [21] Ramos, I. *Logic and OO Databases: a Declarative Approach* Proceedings of the Dexa 90, Springer-Verlag, 1990
- [22] McCabe, F.G., *Logic and Objects*, Prentice-Hall, 1992
- [23] Pastor O. et al. *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*. In Antoni Olivé and Joan Antoni Pastor editors, Proceedings of CAiSE97 conference, pp. 145-158, Berlin, Springer-Verlag, LNCS 1250. June 1997
- [24] Kifer M., *Deductive and Object Data Languages: A Quest for Integration*, Proceedings of the DOOD-96 Conference, Springer-Verlag, 1996
- [25] Carsí J.A.; *A DOOD System for Treating the Schema Evolution Problem*, EDBT'98 Demo Session, VI Intl.Conference on Extending Database Technology, Valencia, March 1998
- [26] Pastor,O.;Insfran,E. ;Pelechano,V. ;García,J. *From Object Oriented Conceptual Modeling to Automated Programming in Java*. Conceptual Modeling – ER'98. Lecture Notes in Computer Science (1507), págs: 183-197, Springer-Verlag, 1998, ISBN: 3-540-65189-6 ; ISSN : 0302-9743, Singapore

Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators

Vânia Maria Ponte Vidal and Bernadette Farias Lóscio

Federal University of Ceará - Computer Science Department
Campus do Pici - Bloco: 910 - 60455-760 Fortaleza-CE – Brazil
{vvidal,berna}@lia.ufc.br

Abstract. Mediator is a facility that supports an integrated view over multiple information sources, and allows for queries to be made against the integrated view. In this paper, we extend the mediator architecture to support updates against the integrated view. Updates expressed against the mediator's integrated view need to be translated into updates of the underlying local databases. We developed algorithms to generate translators for the basic types of mediator update operations. The novel aspects of our algorithms are that the translators are generated based on the correspondence assertions that formally specify the relationships between the mediator schema and the local databases schemas. Our formalism allows us to identify, precisely, the situations where the ambiguities can be solved at mediator definition time and the criteria for choosing the most appropriate translator. In this paper, we show that, by using our formalism, it is possible to define rigorously the correct translation for mediator update operations for cases where concepts in the mediator schema are represented differently in the local databases schemas.

1 Introduction

The mediator architecture has been adopted in many projects [16, 2] to provide integrated access to multiple information sources that can be autonomous and heterogeneous. The mediators are facilities that support an integrated view over multiple information sources. A schema for the integrated view is available from the mediator, and queries can be made against that schema. In this work, we investigate the problem of supporting (limited) updates against the mediator's integrated view.

A mediator definition [15] consists of the instance mappings and the update mappings. The instance mappings specify how instances of the local databases are mapped to instances of the corresponding mediator. The update mappings specify how each update operation defined under the mediator should be translated into a sequence of updates defined under the local databases.

The diagram of Figure 1.1 describes the mediator update translation process. The initial states of the local Databases (DBs), $D_{s_1}, \dots, D_{s_n}$, are mapped by the instance mappings σ_M into the corresponding integrated view state D_M ($D_M = \sigma_M(\{D_{s_1}, \dots, D_{s_n}\})$). The user specifies the update u_M against the integrated view state D_M . The update u_M must be translated into a set of updates $\tau_M(u_M) = \{\tau_{s_1}(u_M), \dots, \tau_{s_n}(u_M)\}$ that can be executed on the local DBs, where $\tau_{s_i}(u_M)$ is the subset of updates in $\tau_M(u_M)$ to be

executed in the schema S_i . The update sequence, $\tau_{S_i}(u_M)$, is performed on the local database state D_{S_i} to obtain the new state $D'_{S_i} = \tau_{S_i}(u_M)(D_{S_i})$. Applying the instance mappings σ_M to the new local DBs states $D'_{S_1}, \dots, D'_{S_n}$, we obtain the corresponding new integrated view state $D'_M = \sigma_M(\{D'_{S_1}, \dots, D'_{S_n}\})$.

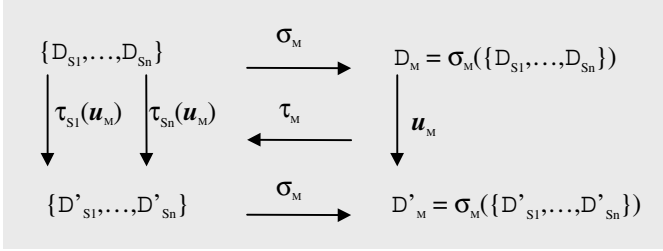


Fig. 1.1. Mediator Update Translation Process

The mediator update problem is similar to that of defining view update translators in centralized databases [1, 5, 7, 8, 9]. In our approach [15], the update mappings consist of a set of translators, one translator for each permitted mediator update operation. A translator is a function that receives an update request and generates a translation for that update. As in Keller's approach, the translators are defined at mediator definition time, and their definitions are stored along with the mediator's specification. We have developed algorithms to generate translators for the basic types of mediator update operations of the Entity and Relationship model. The novel aspects of our algorithms are that the translators are generated based on the correspondence assertions that formally specify the relationships between the mediator schema and the local database schemas. Our formalism allows us to identify, precisely, the situations where the ambiguities can be solved at mediator definition time and the criteria for choosing the most appropriate translator.

One of the difficulties in defining a mediator/view update translator is caused by semantic heterogeneity – the fact that concepts in the mediator/view schema have different representations in the local databases schemas. Our work addresses this problem by extending the ER model to support more general forms of correspondence assertions, which allow the formal specification of the most common types of equivalence of concepts with nonidentical representations. In this paper, we show that, by using our formalism, it is possible to define rigorously the correct translation for mediator update operations even in cases where the mediator schema does not conform to the local databases schemas. This situation has not been properly addressed by other view update algorithms in the literature.

The problem of view update in the ER approach has been investigated by Czejdo[4] and Ling [9]. Ling et al. developed a theory within the framework of the ER approach that characterizes the conditions under which there exists mappings from view updates into updates on the conceptual schema. Those conditions are also considered by our algorithms, and a translator for a given view update operation is generated only in cases where the updatability conditions are satisfied. Our formalism allows us to formally specify the situations where a translator can be defined at mediator definition time. The view update translation algorithms defined in [9] are executed at view

update time and directly translate a valid view update into the corresponding database update, which is not very efficient given the complexity of those algorithms. Our algorithms are used at mediator definition time to automatically generate the translators for a permitted update operation. When the user specifies an update against the mediator, the corresponding translator will be called to generate a translation for the update.

This paper is divided as follows. In Section 2, we present the formalism for the Extended ER model that we use to represent the mediator and local databases schemas. In Section 3, we define the correspondence assertions, which we use to specify the relationships between the mediator schema and the local databases schemas. In Section 4, we discuss our approach to defining translators for mediator update operations and we describe the algorithm that generates correct translators for entity type insertion operations. Finally, in Section 5, we present our conclusions.

2 Terminologies

We illustrate our ideas using an extended ER model [3] to represent the mediator schema and the local databases schemas. An ER schema is a triple $S = (E, R, I)$ where E is a set of entity types, R is a set of relationship types and I is a set of integrity constraints. The types of a schema have attributes that represent structural properties. Attributes can be singlevalued or multivalued. The Domain of an attribute A ($Dom(A)$) specifies the set of values that can be assigned to A .

Next, we introduce some concepts needed for the formal definition of the correspondence assertions used in this paper.

Definition 2.1 (Link): Let X_1 and X_2 be elements in a schema (an element can be an entity type, relationship type or an attribute), X_1-X_2 is a link if:

- (i) X_1 is an attribute of X_2 ; X_1-X_2 is called an *attribute link*;
- (ii) X_1 is an entity type bound by the relationship type X_2 (or vice-versa); X_1-X_2 is called a *relationship link*.

Definition 2.2 (State of a Schema): A state of a schema $S = (E, R, I)$ is a function defined on the sets E and R as follows:

- For any entity type E in E , the extension of E in the state D , $D(E)$, is the set of entities that are instances of E in the state D .
- For any relationship type R of R , linking the entity types $E_1, \dots, E_n$, the extension of R in the state D , $D(R)$, is the set of tuples $\langle e_1, \dots, e_n \rangle$, where $e_i \in D(E_i)$, $1 \leq i \leq n$, that are instances of R in the state D . If $r = \langle e_1, \dots, e_n \rangle \in D(R)$, then we say that “ e_i is linked to r by the link E_i-R ” and “ r is linked to e_i by the link $R-E_i$ ”, $1 \leq i \leq n$. Each relationship link E_i-R has a minimum and maximum cardinalities specifying, respectively, the minimum and maximum number of instances of R to which an instance of E_i can be linked.
- For any attribute A of a type T^1 of S , the value of the attribute A for an instance t of T in the state D , $t.D(A)$, is the set of values in $Dom(A)$ assigned to t in the state D . If

¹ T can be an entity type or a relationship type.

A is a singlevalued attribute, $t.D(A)$ has a single value. If $v \in t.D(A)$, then we say that “ t is linked to v by the attribute link $T-A$ ”.

Definition 2.3 (Path): If $X_1, X_2, \dots, X_n$ are elements of a schema such that $\forall i \in \{1, 2, \dots, n-1\}$, X_i-X_{i+1} is a link, then $X_1-X_2-\dots-X_n$ is a “Path of X_1 ”. An instance x_i of X_i is linked to an instance x_n of X_n in the state D , if there exists $x_2 \in D(X_2), \dots, x_{n-1} \in D(X_{n-1})$ such that, $\forall i \in \{1, 2, \dots, n-1\}$, x_i is linked to x_{i+1} by the link X_i-X_{i+1} .

Definition 2.4 (State of a Path): The value of the path $P = X_1-X_2-\dots-X_n$ for an instance x_i of X_i in the state D , $x_i.D(P)$, is the set of instances of X_n linked to x_i by the path P . P is singlevalued if an instance of X_i can be linked to at most one instance of X_n , otherwise it is multivalued.

In the rest of this paper, we may omit the state D when D is the current state. For example, $e.P$ means the value of the path P for e in the current state.

Definition 2.5 (Functionally Equivalent Types): The types T_i and T_n are functionally equivalent w.r.t. a path $T_1-T_2-\dots-T_n$ iff an instance of T_i can be linked to at most one instance of T_n by the path $T_1-T_2-\dots-T_n$, and T_n can be linked to at most one instance of T_i by the path $T_n-T_{n-1}-\dots-T_1$.

Definition 2.6 (Composed Attribute Path): If $T_1-\dots-T_m-A_1, \dots, T_1-\dots-T_m-A_n$, are singlevalued paths, where $A_1, \dots, A_n$ are singlevalued attributes of a type T_m , then $T_1-\dots-T_m-\{A_1, \dots, A_n\}$ is a path, and its value for an instance t_i of T_i , $t_i.(T_1-\dots-T_m-\{A_1, \dots, A_n\})$, is an ordered collection of values $\{v_1, \dots, v_n\}$, where $v_i = t_i.(T_1-\dots-T_m-A_i)$, $1 \leq i \leq n$.

3 Correspondence Assertions

We use *Correspondence Assertions* to specify the correspondences between components of schemas. Correspondence Assertions, as defined in [13], are special types of integrity constraints used to assert that the semantics of some components in a schema is somehow related to the semantics of some components in another schema. We identify four classes of correspondence assertions: *Type Correspondence Assertions*, *Attribute Correspondence Assertions*, *Path Correspondence Assertions* and *Existence Dependency Assertions*.

3.1 Type Correspondence Assertions

We use Type Correspondence Assertions to express the different kinds of semantics equivalence of types. The types T_1 and T_2 are “*semantically equivalent*” if for any state D there is a 1-1 mapping function F between instances of T_1 and instances of T_2 ($F : D(T_1) \rightarrow D(T_2)$). In general, T_1 and T_2 have a common identifier², which is used as the mapping function. If an instance t_i of T_1 is mapped to the instance t_2 of T_2 ($F(t_i) = t_2$), then t_i and t_2 are “*semantically equivalent*” ($t_i \equiv t_2$), i.e., t_i and t_2 represent the same object of the real world. Table 1 defines the basic kinds of type assertions.

² An identifier of a type T is a set of paths of T whose values identifies only one instance of T .

Table 1. Type Correspondence Assertions

Relation	Notation	Condition for validity in $\mathcal{D}$
<i>Equivalence</i>	$T_1 \equiv T_2$	Exists a bijective function $F: \mathcal{D}(T_1) \rightarrow \mathcal{D}(T_2)$
<i>Difference</i>	$T \equiv T_1 - T_2$	Exists a bijective function $F: \mathcal{D}(T) \rightarrow \mathcal{D}(T_1) - \mathcal{D}(T_2)$
<i>Union</i>	$T \equiv \bigcup_{i=1}^n T_i$	Exists a bijective function $F: \mathcal{D}(T) \rightarrow \bigcup_{i=1}^n \mathcal{D}(T_i)$
<i>Intersection</i>	$T \equiv \bigcap_{i=1}^n T_i$	Exists a bijective function $F: \mathcal{D}(T) \rightarrow \bigcap_{i=1}^n \mathcal{D}(T_i)$
<i>Selection</i> ³	$T_1 \equiv T_2[P]$	Exists a bijective function $F: \mathcal{D}(T_1) \rightarrow \mathcal{D}(T_2[P])$

3.2 Attribute Correspondence Assertions

Attribute Correspondence Assertions specify that attributes of semantically equivalent types are synonyms. This can be formalized as follows:

Definition 3.1: Let A_1 be an attribute of type T_1 and A_2 be an attribute of type T_2 , where types T_1 and T_2 are semantically equivalent. The attribute assertion $A_1 \equiv A_2$ is valid in a state $\mathcal{D}$ if, for any $t_1 \in \mathcal{D}(T_1)$ and $t_2 \in \mathcal{D}(T_2)$, $t_1 \equiv t_2$ implies that $t_1.\mathcal{D}(A_1) = t_2.\mathcal{D}(A_2)$.

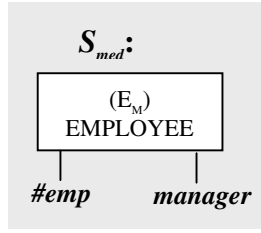
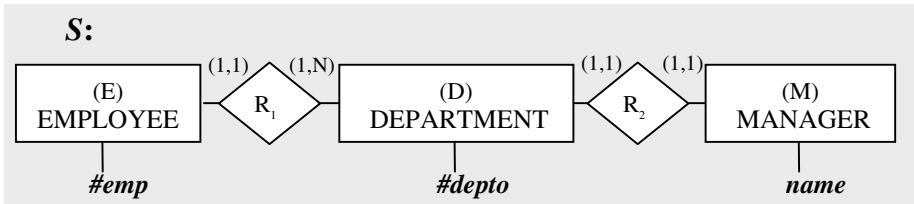
3.3 Path Correspondence Assertions

Path Correspondence Assertions specify that paths of semantically equivalent types are semantically equivalent. Attribute assertions are special types of path assertion. Path assertions can be formalized as follows:

Definition 3.2: Consider the paths $P_1 = X_1 - \dots - X_n$ and $P_2 = Y_1 - \dots - Y_n$, where types X_i and Y_i are semantically equivalent. The path assertion $P_1 \equiv P_2$ is valid in a state $\mathcal{D}$ if, for any $x \in \mathcal{D}(X_1)$ and $y \in \mathcal{D}(Y_1)$, $x \equiv y$ implies that $x.\mathcal{D}(P_1) = y.\mathcal{D}(P_2)$.

The most common types of path assertions found when integrating a mediator schema with local databases schemas are those that specify the derivation path of a derived attribute. Consider for example the mediator schema S_{med} and the local schema S presented in figures 3.1 and 3.2, respectively. The path assertion $E_M\text{-manager} \equiv E - R_1 - D - R_2 - M - \text{name}$ specifies that the value of the attribute *manager* of an employee in S_{med} is derived from the *name* of the manager of the department of that employee in S . The attribute *manager* is a derived attribute since it is not an attribute of *EMPLOYEE* (which is the base type of E_M), but an attribute of the entity type *MANAGER* that is related to *EMPLOYEE* by the path $E - R_1 - D - R_2 - M$.

³ For any type T and predicate P , $\mathcal{D}(T[P]) = \{t \mid t \in \mathcal{D}(T) \wedge P(t) = \text{true}\}$.

Fig. 3.1. Mediator Schema S_{med} Fig. 3.2. Local Schema S

3.4 Existence Dependence Assertions

Existence Dependence Assertions (EDA) [14] are used to express how the existence of instances of a type depends on the existence of instances of other types. As shown in [14] type correspondence assertions are special types of existence dependency assertion. In [10] we define five different types of EDAs. In following, we formally define the type of EDA used in this paper's example.

Definition 3.3: Let $P_{11}, \dots, P_{1n}$ be a singlevalued path of type T_1 and $P_{21}, \dots, P_{2n}$ be a singlevalued path of type T_2 . The EDA $T_1[P_{11}, \dots, P_{1n}] \subseteq T_2[P_{21}, \dots, P_{2n}]$, is valid in a state D if for every $t_1 \in D(T_1)$ there is a $t_2 \in D(T_2)$ such that $t_1.D(P_{1i}) = t_2.D(P_{2i})$ for $1 \leq i \leq n$.

4 Defining Correct Translators for Mediator Update Operations

We propose a methodology for designing of mediators [15] consisting of the following steps: (i) *Mediator schema definition*: The user requirements are analysed and the mediator schema is specified using a high-level data model. (ii) *Mediator schema integration*: The mediator schema is integrated with the local schemas to identify the correspondence assertions that formally specify the relationships between the mediator schema and the local schemas. (iii) *Mediator definition*: Using the mediator schema and the correspondence assertions, the instance mappings and the update mappings are defined.

In our approach, the update mappings consist of a set of translators, one translator for each permitted mediator update operation. A translator is a function that receives an update request and generates a translation for that update. The translators are defined at mediator definition time, and their definitions are stored along with the mediator's specification. We have developed algorithms to generate translators for the basic types of mediator update operations (Entity/Relationship type Insertion, Entity/Relationship type Deletion and Attribute Modification).

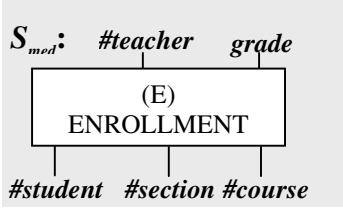
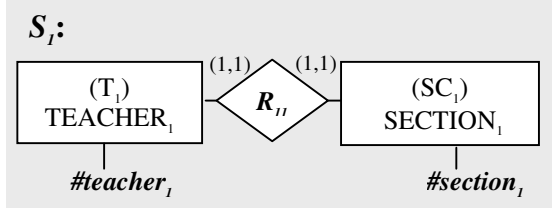
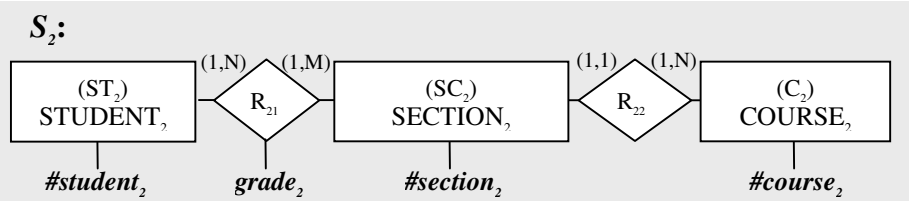
The Algorithm *A1*, presented in the Appendix A, generates translators for entity type insertion operations. In our algorithms, the translators are generated based on the correspondence assertions that formally specify the relationships between the mediator schema and the local schemas. The mediator correspondence assertions are treated as integrity constraints that should be preserved by the mediator update operations. In our approach, we consider as if the mediator update were executed directly in the virtual integrated view, so that, the translation consists of a sequence of updates that must be performed in the local databases to make them "synchronized" with the new Integrated View. Therefore, the problem of defining a translator for a given mediator update operation consists in determining the sequences of local databases updates that are required for the maintenance of the mediator correspondence assertions which are relevant⁴ to that operation.

One of the difficulties associated with the mediator update problem arises when there are multiple possible translations, but only one must be chosen. This is the case of a correspondence assertion where there exists, for a certain type of update, more than one possible way of preserving consistency. In our approach, for those correspondence assertions for which several alternative corrections are possible, if the ambiguities can be solved at mediator definition time, the designer should extend the declaration of the assertion with additional information to indicate which option should be used. Those information constitute the "*update semantics*" of the correspondence assertion. It is important to notice, that our algorithms always use the alternative that minimizes the view's side effect; therefore, the definition of the update semantics is needed only in cases where the alternatives do not offer any difference in terms of the view's side effect. Consider for example, the correspondence assertion $T_M \equiv T_1 \cup T_2$. The insertion of an instance in T_M , requires the insertion of a semantically equivalent instance in one of the base types T_1 or T_2 . The update semantics of the assertion should specify which entity type was chosen by the designer. The update semantics of a correspondence assertion define an update policy, and they are used by our algorithms to define the correct actions for the maintenance of the mediator correspondence assertions. Our formalism allows us to identify, precisely, the situations where the ambiguities can be solved at mediator definition time.

In this section, we use the mediator schema S_{med} presented in Figure 4.1 as an example. S_{med} integrates information of two databases whose schemas S_1 and S_2 are presented in figures 4.2 and 4.3, respectively. Figure 4.4 shows the "*Add_ENROLLMENT*" translator, generated by Algorithm *A1*, which is used for translating the requests to add an instance of the mediator type **ENROLLMENT**. When the mediator receives an update request to add an instance " e_M " into the entity type **ENROLLMENT** then the "*Add_ENROLLMENT*" translator will be called with

⁴ A correspondence assertion is relevant to an operation if it can be violated by the operation.

the entity specification e_M passed as parameter. e_M is an entity specification of **ENROLLMENT** given by: $\{ \langle \#student: v_{\#student} \rangle, \langle \#section: v_{\#section} \rangle, \langle \#course: v_{\#course} \rangle, \langle \#teacher: v_{\#teacher} \rangle, \langle \#grade: v_{\#grade} \rangle \}$, where $v_{\#student}$ is the value of the attribute **#student** of e_M , $v_{\#section}$ is the value of the attribute **#section** of e_M , and so on. An entity specification of an entity type E only specifies the values for the single-valued attributes of E .

Fig. 4.1. Mediator Schema S_{med} Fig. 4.2. Local Schema S_1 Fig. 4.3. Local Schema S_2

As mentioned before, the translator for a mediator update operation determines the sequences of local database updates that are required for maintenance of the operation's relevant assertions. In our example, the relevant assertions for insertion in the mediator entity type **ENROLLMENT** are:

- The type correspondence assertion:
 $\Psi_1: E \equiv R_{21}$
- The attribute correspondence assertion:
 $\Psi_2: \text{grade} \equiv \text{grade}_2$
- The path correspondence assertions:
 $\Psi_3: E - \#student \equiv R_{21} - ST_2 - \#student_2$
 $\Psi_4: E - \#section \equiv R_{21} - SC_2 - \#section_2$
 $\Psi_5: E - \#course \equiv R_{21} - SC_2 - R_{22} - C_2 - \#course_2$
- The existence dependency assertion:
 $\Psi_6: E[E - \#teacher, E - \#section] \subseteq R_{11}[R_{11} - T_1 - \#teacher_1, R_{11} - SC_1 - \#section_1]$

In the following, we discuss the actions required for maintaining those assertions, as defined in Algorithm A1.

Step 1: Maintenance of Type Correspondence Assertions

Step 1 of Algorithm A1 defines the actions required for maintenance of the type assertions. As discussed in Section 3, the type correspondence assertions are used to express the different kinds of semantics equivalence of types. If a mediator type T_M is related to a local database type T by any of the type correspondence assertions defined in table 1, then we say that T is a *base type* of T_M .

Lines 1 and 2 of the “**Add_ENROLLMENT**” translator were generated by the Step1 of Algorithm A1 and define the updates required in the local databases for maintenance of the type assertion Ψ_1 . The assertion Ψ_1 specifies that the entity type **ENROLLMENT** and the relationship type R_{21} are semantically equivalent, therefore the insertion of an instance e_M into **ENROLLMENT** requires the insertion of an instance r_{21} into R_{21} , that is semantically equivalent to e_M ($r_{21} \equiv e_M$, i.e., r_{21} and e_M represent the same object of the real world). The new relationship r_{21} is created using the instance constructor “**create_** R_{21} **_from_****ENROLLMENT**” presented in figure 4.5, which is a function that receives the entity specification e_M of **ENROLLMENT** and returns the relationship specification r_{21} of R_{21} , such that $r_{21} \equiv e_M$.

The constructor “**create_** R_{21} **_from_****ENROLLMENT**” was generated by Algorithm A2 presented in the Appendix A. That algorithm generates constructors that transforms an entity specification into a semantically equivalent relationship specification. In the same way as the translators, the constructors are generated based on the correspondence assertions that formally specify the relationships between the mediator schema and the local schemas. Due to that formalism, we could define algorithms to automatically generate constructors, such as “**create_** R_{21} **_from_****ENROLLMENT**,” that correctly defines the mappings between instances of types that have different structures. In [10] we present algorithms to generate constructors for other types of instance mappings.

The Algorithm A2 generates the constructor to transform an entity specification e_M of the mediator entity type E_M into a relationship specification r of the base type R . The algorithm consists of two steps: the first step determines all the entities participating in the relationship r , and the second step determines the values of the attributes of r . Then, the relationship specification $\{ \langle E_1; e_1 \rangle \dots \langle E_k; e_k \rangle \langle A_1; v_{A_1} \rangle \dots \langle A_p; v_{A_p} \rangle \}$, where $E_1, \dots, E_k$ are the participating entity types of R and $A_1, \dots, A_p$ are attributes of R , is returned.

Lines 1 and 2 of the constructor “**create_** R_{21} **_from_****ENROLLMENT**”, generated by the Step 1 of Algorithm A2, determine all the entities participating in the relationship r_{21} of R_{21} such that $r_{21} \equiv e_M$. Line 1, generated based on the path assertion Ψ_3 (Case 1.1), determines the instance of $STUDENT_2$ participating in r_{21} . Ψ_3 specifies that the attribute **#student** of **ENROLLMENT** and the path $R_{21} - ST_2 - \#student_2$ are semantically equivalent. Therefore we have that $e_M.(E - \#student) = r_{21}.(R_{21} - ST_2 - \#student_2)$. Since $\{\#student_2\}$ is an identifier of $STUDENT_2$, the instance of $STUDENT_2$ is the entity st such that $st.\#student_2 = v_{\#student}$. Line 2, generated based on the path assertion Ψ_4 (Case 1.1), determines the instance of $SECTION_2$ participating in r_{21} . Ψ_4 specifies that the attribute **#section** of **ENROLLMENT** and the path $R_{21} - SC_2 - \#section_2$ are semantically equivalent. Therefore we have that $e_M.(E - \#section) = r_{21}.(R_{21} - SC_2 - \#section_2)$. Since $\{\#section_2\}$ is an identifier of $SECTION_2$,

the instance of $SECTION_2$ is the entity sc such that $sc.\#section_2 = v_{\#section}$. In our algorithms, when a searched object is not found, an exception will be raised and it is handled by rollingback the transaction. Note that in a case where it is not possible to identify all the entities participating in the relationship, the constructor cannot be defined, and, consequently, the translator cannot be defined.

Line 3 of the constructor “ $create\_R_{21\_from\_ENROLLMENT}$ ”, generated by the Step2 of Algorithm A2, determines the value of the attribute $grade_2$ of the relationship r_{21} . Line 3 was generated based on the path assertion Ψ_2 that specifies that the attributes $grade$ and $grade_2$ are semantically equivalent. Therefore we have that $e_M.\#grade = r_{21}.\#grade_2$, thus $r_{21}.\#grade_2 = v_{\#grade}$.

```

Add_ENROLLMENT ( $e_M$ )
{
/* Translates the requests to add  $e_M$  into ENROLLMENT.  $e_M$  is an entity
specification given by: {<#student:  $v_{\#student}$ > <#section:  $v_{\#section}$ > <#course:
 $v_{\#course}$ > <#grade:  $v_{\#grade}$ > <#teacher:  $v_{\#teacher}$ >} */
/* Actions required for maintenance of  $\Psi_1$ */
1.  $r_{21} := create\_R_{21\_from\_ENROLLMENT}(e_M)$ ;
2. Add  $r_{21}$  into  $R_{21}$ ;
/* Actions required for maintenance of  $\Psi_5$ */
3. Obtain  $r_{21}$  in  $R_{21}$  such that  $r_{21} \equiv e_M$ ;
4. If  $v_{\#course} \neq \text{'NULL'}$  and  $r_{21}.(R_{21} - \dots - C_2 - \#course_2) \neq \text{' $v_{\#course}$ '}$  then Rollback;
/* Actions required for maintenance of  $\Psi_6$ */
5. If there not exists  $r_{11}$  in  $R_{11}$  such that
6.  $r_{11}.(R_{11} - T_1 - \#teacher_1) = v_{\#teacher}$  and  $r_{11}.(R_{11} - SC_1 - \#section_1) = v_{\#section}$ ;
7. then Rollback;
}

```

Fig. 4.4. The Translator “Add_ENROLLMENT”

```

create_R21_from_ENROLLMENT ( $e_M$ )
{
/* Creates the relationship specification  $r_{21} = \{<STUDENT_2: st> <SECTION_2: sc>
<grade_2:  $v_{\#grade_2}$ >> of  $R_{21}$ , such that  $r_{21}$  is semantically equivalent to the entity
specification  $e_M = \{<#student:  $v_{\#student}$ > <#section:  $v_{\#section}$ > <#course:  $v_{\#course}$ >
<#grade:  $v_{\#grade}$ > <#teacher:  $v_{\#teacher}$ >> of ENROLLMENT */
1. Obtain  $st$  in  $STUDENT_2$ , such that  $st.\#student_2 = v_{\#student}$ ;
2. Obtain  $sc$  in  $SECTION_2$ , such that  $sc.\#section_2 = v_{\#section}$ ;
3.  $v_{\#grade_2} := v_{\#grade}$ ;
4. Return {<STUDENT2:  $st$ > <SECTION2:  $sc$ > <grade2:  $v_{\#grade_2}$ >};
}$$ 
```

Fig. 4.5. The Constructor “create_R₂₁_from_ENROLLMENT”

Step 2: Maintenance of the remaining Path Correspondence Assertions

Step 2 of Algorithm A1 defines the actions required for maintenance of the path correspondence assertions not yet considered at Step 1. Those path assertions specify the derivation path of a mediator derived attribute. In our approach, a value can be assigned to a mediator derived attribute only in a very restricted situations (Case 2.1 of Algorithm A1). This is the case of the derived attribute **manager** of the mediator in figure 3.1. The path assertion $E_M - \text{manager} \equiv E - R_1 - D - R_2 - M - \text{name}$ specifies that the value of the attribute **manager** of an employee in E_M is derived from the **name** of the manager of the department of that employee in E . As specified in Case 2.1 of Algorithm A1, in order to assign a value v_{manager} to an employee in E_M , we have to add a relationship in R_1 associating that employee with the department which has a manager whose name is v_{manager} . In [10] we present an algorithm to generate translator for mediator attribute modification and we discuss the problem of modifying derived attribute.

Lines 3 and 4 of the translator “Add_ENROLLMENT” were generated by Step 2 of Algorithm A1 and define the actions required in the local databases for maintenance of the path assertion Ψ_5 . The assertion Ψ_5 specifies that the attribute **#course** and the path $R_{21} - SC_2 - R_{22} - C_2 - \text{#course}_2$ are semantically equivalent. Since R_{21} is not an entity type, Ψ_5 does not satisfy the conditions of Case 2.1 of Algorithm A1 and a value cannot be assigned to **#course**. In fact, it is most probable that the relationship r_{21} , previously inserted in R_{21} , already has a defined value for the derivation path of **#course**. So, if a value is assigned to **#course** and the value assigned is not consistent with the value of its derivation path ($r_{21}(R_{21} - SC_2 - R_{22} - C_2 - \text{#course}_2) \neq v_{\text{#course}}$), then rollback the transaction. The same procedure is applied in case the derivation path has not yet a defined value; for we think, it would not be correct to allow that an insertion in the mediator type E can cause an insertion in R_{22} , since R_{22} is not a relationship of a base type of E .

Step 3: Maintenance of Existence Dependency Assertions

Step 3 of Algorithm A1 defines the actions required for maintenance of the existence dependency assertions. Lines 5 to 7 of the translator “Add_ENROLLMENT” were generated by the Step 3 of Algorithm A1 and define the actions required in the local databases for maintenance of the existence dependency assertion Ψ_6 . The assertion Ψ_6 specifies that the existence of an entity e_M in **ENROLLMENT** requires the existence of a relationship r_{11} in R_{11} where $e_M(E - \text{#teacher}) = r_{11}(R_{11} - T_1 - \text{#teacher}_1)$ and $e_M(E - \text{#section}) = r_{11}(R_{11} - SC_1 - \text{#section}_1)$. Therefore, if doesn't exist r_{11} in R_{11} such that $r_{11}(R_{11} - T_1 - \text{#teacher}_1) = v_{\text{#teacher}}$ and $r_{11}(R_{11} - SC_1 - \text{#section}_1) = v_{\text{#section}}$, rollback the transaction.

5 Conclusions

In this paper, we extended the mediator architecture to support updates against the mediator integrated view. In our approach, we generate a translator for each permitted mediator update operation. The translators are defined at mediator definition time, and their definitions are stored along with the mediator specification. We developed

algorithms to automatically generate translators for the basic types of mediator update operations. In our algorithms, the translators are generated based on the correspondence assertions that formally specify the relationships between the mediator schema and the local schemas. The mediator correspondence assertions are treated as integrity constraints that should be preserved by the mediator update operations. Therefore, the problem of defining a translator for a given mediator update operation consists in determining the sequences of local databases updates that are required for the maintenance of the mediator correspondence assertions which are relevant to that operation.

In our approach, ambiguities occurs when there exists, for a certain type of update, more than one possible way of preserving the consistency of a given correspondence assertion. For those correspondence assertions for which several alternative corrections are possible, if the ambiguities can be solved at mediator definition time, the designer should extend the declaration of the assertion with additional information to indicate which option should be used (update semantics). The update semantics of a correspondence assertion define an update policy, and they are used by our algorithms to define the correct actions for the maintenance of the mediator correspondence assertion. Our formalism allows us to identify, precisely, the situations where the ambiguities can be solved at mediator definition time and the criteria for choosing the most appropriate translator.

In this paper, we presented the Algorithm *A1* that generates translators for entity type insertion operations. We showed that, due to our formalism, we could define algorithms to automatically generate translators that define correct translation for mediator update operations. This is so, even in cases where the mediator are represented differently in the local database schemas (semantic heterogeneity). To illustrate that, we showed an example where we used the Algorithm *A1* to generate a translator that correctly defines the mappings from insertions in a mediator entity type into insertions in a base relationship type.

It is important to notice that ours is intended to be a general approach for solving the whole problem of mediator/view update, and our algorithms can be easily adapted to other data models. One advantage of using the ER model is that, because of its semantic power, it is possible to have a better understanding of the semantics associated with a given view update translation. Therefore, the semantics power of the ER model combined with our formalism allows us to formally justify that the translators generated by our algorithms are correct.

References

1. Bancilhon, F., Spyrtos, N.: Updates Semantics and Relational View. *ACM Transactions on Database Systems*, Vol. 6, no.4, (1981) 557-575
2. Chawathe, S., Garcia Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of Info. Processing Society of Japan* (1994)
3. Chen, P.P.: The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, Vol 1, no.1 (1976) 166-192
4. Czejdo, B., Embley, D.W. and Rusinkiewicz, M.: View Updates for an Extended Entity-Relationship Model. *Information Sciences*, no.62 (1992) 41-62

5. Dayal, U., Bernstein, P. A.: On the Correct Translation of Update Operations on Relational Views. ACM Transactions on Database Systems, Vol. 7, no.3 (1982) 381-416.
6. Hull, R: Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In Proceedings of ACM Symp. on Principles of Database, Tucson Arizona, USA (1997) 51-61
7. Keller, A. M.: The Role of Semantics in Translating View Updates. IEEE Computer, Vol.19, no.1 (1986) 63-73
8. Larson, J.A., Sheth, A. P: Updating Relational Views using Knowledge at View Definition and View Update Time. Information Systems, Vol.16, no.2 (1991) 145-168
9. Ling, T.W., Lee, M.L.: View Update in Entity-Relationship Approach. Data & Knowledge Engineering, Vol.19, no.2. (1996) 135-169
10. Lóscio, B.F.: Atualização de Múltiplas Bases de Dados através de Mediadores. M.S. thesis, Computer Science Department, University Federal of Ceará (1998)
11. Navathe, S. B., Savasere, A.: A Schema Integration Facility using Object-Oriented Model. Object Oriented Multidatabase Systems, Prentice Hall (1996)
12. Papakonstantinou, Y., Garcia-Molina, H., Ullman, J.: MedMaker: A Mediation System Based on Declarative Specifications. In Proceedings of International Conference on Data Engineering (1996)
13. Spaccapietra, S., Parent, C.: View Integration: A Step Forward in Solving Structural Conflicts. IEEE Trans. on Knowledge and Data Engineering, Vol. 6, no.2 (1994) 258-274
14. Vidal, V. M. P., Winslett, M.: A Rigorous Approach to Schema Restructuring. In Proceedings of 14Th International Conference on Object-Oriented Entity-Relationship Conference, Gold Coast, Australia, December (1995)
15. Vidal, V. M. P. & Lóscio, B.F.: Especificação de Mediadores para Acesso e Atualização de Múltiplas Bases de Dados. In Proceedings of XII Simpósio Brasileiro de Banco de Dados, Fortaleza Ceará, Brazil, Outubro (1997)
16. Widerhold, G: Mediators in the Architecture of Future Information Systems. IEEE Computer, Vol.25, no.3 (1992) 38-49
17. Wiederhold, G., Genesereth, M.: The Basis for Mediation, IEEE Expert, May (1996)

Appendix A

Algorithm A1: Generates the translator τ for the insertion of an entity e_m into the mediator entity type E_m , where e_m is an entity specification of E_m given by: $\{ \langle A_i, v_{A_i} \rangle > \langle A_2, v_{A_2} \rangle > \dots \langle A_n, v_{A_n} \rangle \}$ where v_{A_i} is the value of the attribute A_i of e_m , v_{A_2} is the value of the attribute A_2 of e_m , and so on.

{

$\tau := \emptyset$; /* Initialization of the translator. The translator consists of a set of local database operations */

Step 1: Maintenance of Type Correspondence Assertions (TCA)

/* Case 1.1 */

For each TCA $E_m \equiv T$, where T is a type do:

$\tau := \tau \cup \{ \langle t := \text{create\_T\_from\_E}_m(e_m);$
Add t into $T; \rangle \}$

/* Case 1.2 */

```

For each TCA  $E_M \equiv T[P]$ , where  $T$  is a type and  $P$  is a
predicate do:
 $\tau := \tau \cup \{ \langle t := \text{create\_}T\_from\_E_M(e_M);$ 
    Add  $t$  into  $T$ ;  $\rangle \}$ 
/* Case 1.3 */
For each TCA  $T \equiv E_M[P]$ , where  $T$  is a type and  $P$  is a
predicate do:
 $\tau := \tau \cup \{ \langle \text{If } P(e_M) = \text{'true' then}$ 
     $t := \text{create\_}T\_from\_E_M(e_M);$ 
    Add  $t$  into  $T$ ;  $\rangle \}$ 
/* Case 1.4 */
For each TCA  $E_M \equiv T_1 - T_2$ , where  $T_1$  and  $T_2$  are types do:
 $\tau := \tau \cup \{ \langle \text{If there exists } t_1 \text{ in } T_1 \text{ such that } t_1 \equiv e_M \text{ then}$ 
    If there exists  $t_2$  in  $T_2$  such that  $t_2 \equiv e_M$ 
    then Delete  $t_2$  in  $T_2$ ;
    else  $t_1 := \text{create\_}T_1\_from\_E_M(e_M);$ 
    If there exists  $t_2$  in  $T_2$  such that  $t_2 \equiv e_M$ 
    then Delete  $t_2$  in  $T_2$ ;
    Add  $t_1$  into  $T_1$ ;  $\rangle \}$ 
/* Case 1.5 */
For each TCA  $E_M \equiv \bigcup_{j=1}^k T_j$ , where  $T_1, \dots, T_k$  are types and
 $T_i$  is the type where the insertion should be made
(update semantics) do:
 $\tau := \tau \cup \{ \langle t := \text{create\_}T_i\_from\_E_M(e_M);$ 
    Add  $t$  into  $T_i$ ;  $\rangle \}$ 
/* Case 1.6 */
For each TCA  $E_M \equiv \bigcap_{j=1}^k T_j$ , where  $T_1, \dots, T_k$  are types do:
 $\tau := \tau \cup \{ \langle \text{For each } T_j \text{ do:}$ 
     $t := \text{create\_}T_j\_from\_E_M(e_M);$ 
    Add  $t$  into  $T_j$ ;  $\rangle \}$ 

```

Step 2: Maintenance of the remaining Path
Correspondence Assertions (PCA)

```

/* Case 2.1 */
For each PCA  $E_M - \{A_{i_1}, \dots, A_{i_p}\} \equiv E_1 - R_1 - E_2 \dots - E_n - \{B_1, \dots, B_p\}$ 
where  $1 \leq i_k \leq n$  for  $1 \leq k \leq p$ ,  $E_1$  is an entity type,  $E_2$ 
and  $E_n$  are functionally equivalent ( $E_2 \leftrightarrow E_n$ ) w.r.t the
path  $E_1 - R_1 - E_2 \dots - E_n$ , and  $\{B_1, \dots, B_p\}$  is an identifier of
 $E_n$ , do:
 $\tau := \tau \cup \{ \langle \text{If } \forall A_{i_k} \neq \text{'NULL' then}$ 
    Obtain  $e_1$  in  $E_1$  such that  $e_1 \equiv e_M$ ;
    Obtain  $e_2$  in  $E_2$  such that  $e_2.(E_2 \dots E_n - B_k)$ 
    =  $\forall A_{i_k}$ ,  $1 \leq k \leq p$ ;
    Add  $\langle R_1: \langle E_1: e_1 \rangle \langle E_2: e_2 \rangle \rangle$  into  $R_1$ ;  $\rangle \}$ 

/* Case 2.2 */
For each remaining PCA  $E_M - A_i \equiv T_1 \dots T_n - B$ , where  $T_1, \dots, T_n$ 

```


are types, do:

```

 $\tau := \tau \cup \{ \langle \text{Obtain } t_i \text{ in } T_i \text{ such that } t_i \equiv e_m; \\ \text{If } v_{A_{i_1}} \neq \text{'NULL'} \text{ and } t_i.(T_{i_1} - \dots - T_{i_n} - B) \neq v_{A_{i_1}} \text{ then} \\ \text{Rollback}; \rangle \}$ 

```

Step 3: Maintenance of Existence Dependency
Assertions (EDA)

/* Case 3.1 */

For each EDA $E_M[P_{m_1}, \dots, P_{m_n}] \subseteq T[P_1, \dots, P_n]$, where T is a type and $\{P_1, \dots, P_n\}$ is an identifier of T do:

```

 $\tau := \tau \cup \{ \langle \text{If there not exists } t \text{ in } T \text{ such that } t.P_i = \\ e_m.P_{m_i}, 1 \leq i \leq n, \text{ then} \\ \text{Rollback}; \rangle \}$  /* It is not possible to make the update */

```

/* Case 3.2 */

If there exists EDA $T[P_1, \dots, P_n] \equiv E_M[P_{m_1}, \dots, P_{m_n}]$, where T is a type $\{P_1, \dots, P_n\}$ is an identifier of T then:

Return ($\emptyset$) /*It is not possible to define the translator */

Return (τ);

/* End of the Algorithm A1 */

Algorithm A2: Generates the constructor τ to transform an entity specification e_M of the mediator entity type E_M into a relationship specification r of the base type R . The entity specification e_M is given by: $\{ \langle A_1: v_{A_1} \rangle \langle A_2: v_{A_2} \rangle \dots \langle A_n: v_{A_n} \rangle \}$, where $A_1, \dots, A_n$ are attributes of E_M , and the relationship specification r is given by: $\{ \langle E_i: e_i \rangle \dots \langle E_k: e_k \rangle \langle D_i: v_{D_i} \rangle \dots \langle D_r: v_{D_r} \rangle \}$, where $E_1, \dots, E_k$ are the participating entity types of R and $D_1, \dots, D_r$ are attributes of R .

{

$\tau := \emptyset$; /* Initialization of the constructor */

Step 1: Determine all the entities participating in the relationship r

For each participating entity type E_i of R do:

/*Case 1.1*/

If there exists a PCA $E_M - \{A_{i_1}, \dots, A_{i_p}\} \equiv R - E_i - \{B_1, \dots, B_p\}$,

where $1 \leq i_k \leq n$ for $1 \leq k \leq p$, and $\{B_1, \dots, B_p\}$ is an identifier of E_i then

```

 $\tau := \tau \cup \{ \langle \text{Obtain } e_i \text{ from } E_i, \text{ such that } e_i.B_k = v_{A_{i_k}}, \\ 1 \leq k \leq p \rangle; \}$ 

```

else

```
/* Case 1.2 */
```

If there exists a PCA $\mathbf{E}_M - \{\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_p}\} \equiv \mathbf{R} - \mathbf{E}_1 - \dots - \mathbf{F} - \{\mathbf{B}_1, \dots, \mathbf{B}_p\}$, where $1 \leq i_k \leq n$ for $1 \leq k \leq p$, $\mathbf{E}_i$ and $\mathbf{F}$ are functionally equivalent ($\mathbf{E}_i \leftrightarrow \mathbf{F}$) w.r.t the path $\mathbf{E}_1 - \dots - \mathbf{F}$, and $\{\mathbf{B}_1, \dots, \mathbf{B}_p\}$ is an identifier of $\mathbf{F}$ then

```

     $\tau := \tau \cup \{ \text{Obtain } \mathbf{f} \text{ in } \mathbf{F}, \text{ such that } \mathbf{f} \cdot \mathbf{B}_k = \mathbf{V}_{\mathbf{A}_{i_k}},$ 
       $1 \leq k \leq p;$ 
      Obtain  $\mathbf{e}_i$  in  $\mathbf{E}_i$ , such that  $\mathbf{e}_i = \mathbf{f} \cdot (\mathbf{F} \dots \mathbf{E}_i); > \}$ 
else Return( $\emptyset$ ); /* It is not possible to define the
      constructor */

```

Step 2: Determine the values of the attributes of **r**

For each attribute D_i of R do:

If there exists a TCA $\mathbf{E}_m - \mathbf{A}_j \equiv \mathbf{R} - \mathbf{D}_i$, $1 \leq j \leq n$, then

$$\tau := \tau \cup \{ \langle \mathbf{V}_{D_i} := \mathbf{V}_{A_j}; \rangle \}$$

$$\text{else } \tau := \tau \cup \{ \langle \mathbf{V}_{D_i} := \text{'NULL'}; \rangle \}$$
$$\tau := \tau \cup \langle \text{Return } (\{ \langle \mathbf{E}_1: \mathbf{e}_1 \rangle \dots \langle \mathbf{E}_k: \mathbf{e}_k \rangle \langle \mathbf{D}_1: \mathbf{v}_{D_1} \rangle \dots \langle \mathbf{D}_r: \mathbf{v}_{D_r} \rangle \}) \rangle$$

```
Return ( $\tau$ );
}/* End of the Algorithm A2 */
```

A Method for Requirements Elicitation and Formal Specification

Maritta Heisel¹ and Jeanine Souquière²

¹ Institut für Verteilte Systeme, Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg, D-39016 Magdeburg, Germany
email: heisel@cs.uni-magdeburg.de

² LORIA—Université
Nancy2, B.P. 239 Bâtiment LORIA, F-54506, Vandœuvre-les-Nancy, France
souquier@loria.fr

Abstract. We propose a method for the elicitation and the expression of requirements. The requirements are then transformed in a systematic way into a formal specification. The approach – which distinguishes between requirements and specifications – gives methodological support for requirements elicitation and specification development. It avoids introducing new notations but builds on known techniques.

1 Introduction

The usefulness of formal specification is more and more accepted by researchers and practical software engineers. Consequently, this subject is treated in the majority of software engineering text books, and many formal specification languages have been developed.

But formal specification techniques still suffer from two drawbacks. First, research spends more effort to develop new languages than to provide methodological guidance for using existing ones. Often, users of formal techniques are left alone with a formalism for which no explicit methodology has been developed.

Second, formal specification techniques are not well integrated with the analysis phase of software engineering. The starting point from which the development of a formal specification should begin is not well elaborated. Often, formal specification begins with a very short description of the system to be implemented, and detail is added during the development of the formal specification. Such a procedure does not adequately take into account the need to thoroughly analyze the system to be implemented and the environment in which it will operate *before* a detailed specification is developed. The elicitation of the requirements for the system and their transformation into a formal specification are separate tasks.

This paper treats both of these issues. It introduces an explicit requirements elicitation phase, the result of which is an adequate starting point for the development of the formal specification. Furthermore, it provides substantial methodological guidance for requirements elicitation as well as for specification acquisition. The different phases provide feedback to one another: not only is the

specification based on the requirements, but the specification phase may also reveal omissions and errors in the requirements.

Our approach opts for a formal expression of requirements. After some informal brainstorming steps, the requirements are formalized as constraints on sequences of events that can happen or operations that can be invoked in the context of the system. Thus, the transition from informal to formal means of expression is performed very early in the software development process. This has the advantage that requirements can be analyzed, e.g., for consistency, and that it is possible to define a formal notion of correctness of a specification with respect to given requirements.

In the following, we describe the methods for requirements elicitation (Section 2) and specification acquisition (Section 3), which are illustrated by an example, an automatic teller machine. Section 4 discusses the state of the art, and a summary of our contributions concludes the paper.

2 Requirements Elicitation

Our approach to requirements engineering is inspired by the work of Jackson and Zave [10,17] and by the first steps of object oriented methods. The starting point is a brainstorming process where the application domain and the requirements are described in natural language. This informal description is then transformed into a formal representation.

Our approach is suitable for a large class of systems: transformational as well as reactive systems. Transformational systems perform data transformations. They offer a set of system operations that can be invoked by the user. Reactive systems, on the other hand, have to react to events that happen in their environment.

The difference between requirements and a specification is that requirements refer to the entire system to be realized, whereas a specification refers only to the part of the system to be implemented by software. To express requirements formally, we use *traces* of the system, i.e., sequences of events that happen in certain states and at a certain time. For transformational systems, events can be identified, too, namely the invocation and the termination of the system operations. In this way, systems that are partially reactive and partially transformational can also be treated.

2.1 Agenda for Requirements Elicitation

Requirements elicitation is performed in six steps. To express our method, we use the *agenda* concept [7].

An agenda is a list of steps to be performed when carrying out some task in the context of software engineering. The result of the task will be a document expressed in some language. Agendas contain informal descriptions of the steps, which may depend on each other. Usually, they will have to be repeated to achieve the goal, because later steps will reveal errors and omissions in earlier steps. Agendas are presented as tables, see Table 1.

Agendas are not only a means to guide software development activities. They also support quality assurance because the steps may have validation conditions

associated with them. These validation conditions state necessary semantic conditions that the developed artifact must fulfill in order to serve its purpose properly. Validation conditions that can in principle be checked mechanically are marked $\vdash$, validation conditions that can only be checked informally are marked $\circ$.

No	Step	Validation Conditions
1	Fix the domain vocabulary.	$\circ$ The vocabulary must contain exactly the notions occurring in the facts, assumptions, requirements, operations and events.
2	State the facts, assumptions and requirements concerning the system in natural language.	
3	List the possible system operations that can be invoked by the users, together with their input and output parameters.	
4	List all relevant events that can happen in connection with the system, together with their parameters.	
5	Classify the events.	$\vdash$ There must not be any events controlled by the software system and not shared with the environment.
6	Formalize facts, assumptions and requirements as constraints on the admissible traces of system events.	$\circ$ Each requirement of Step 2 must be expressed. $\vdash$ The constraints must be consistent. $\vdash$ For each predicate introduced, the events that modify it must be shared with the software system.

Table 1. Agenda for requirements elicitation

The starting point of the requirements elicitation phase is an informal requirements document provided by the customer. The goal of the first five steps of our method is to understand the problem, to fix the domain vocabulary, and to state the requirements more precisely. They usually will need communication with the customer. The result of these brainstorming steps forms the starting point for the formal expression of the requirements.

We now explain the steps of the agenda one by one.

Step 1. Fix the domain vocabulary.

In informal requirements documents, the used vocabulary is often ambiguous. Several names may be used to refer to the same thing or concept, and even the same word may be used to refer to different concepts. In this step, the domain vocabulary should be fixed in such a way that for each concept exactly one name is used.

To represent the domain vocabulary, we use a notation similar to entity-relationship diagrams [1].

Step 2. State the facts, assumptions and requirements concerning the system.

It does not suffice to just state requirements for the system. Often, facts and assumptions must be introduced to make the requirements satisfiable. *Facts* express phenomena that always hold in the application domain, regardless of the implementation of the software system. These are often hardware properties, e.g., that a card reader can hold at most hold one card. Other requirements cannot be enforced because, e.g., human users might violate regulations. In such a case, it may be possible that the system can fulfill its purpose properly only under the condition that users behave as required. Such conditions are expressed as *assumptions*, which can be used to show that the software system is correctly implemented. *Requirements* constrain the specification and implementation of the software system. It must be *demonstrated* that they are fulfilled, once the specification and the implementation are finished. In the specification phase, it must be shown that the modeling of the software system takes the facts into account and that the requirements are fulfilled, whereas the assumptions can be taken for granted.

If it is not already done in the initial requirements document, the facts, assumptions and requirements should be stated as *fragments* as small as possible. Such fragments can correspond to (parts of) independent scenarios of system behavior. For reasons of traceability, each fragment should be given a unique name or number.

Step 3. List the possible system operations that can be invoked by the users, together with their input and output parameters.

This step is concerned with the non-reactive part of the system to be described. For purely reactive systems, it can be empty. System operations are usually independent of the physical components of the system, but refer to the information that is stored in the part of the system to be realized by software.

With each system operation *Op*, the events *OpInvocation* and *OpTermination* are associated, where *OpInvocation* is controlled by the environment and shared with the software system, whereas the *OpTermination* is controlled by the software system and shared with the environment.

Step 4. List all relevant events that can happen in connection with the system, together with their parameters.

This step only concerns the reactive part of the system. For purely transformational systems, the set of events may be empty. An event usually has a relation with a physical part of the system, such as a button or a light.

Step 5. Classify the events.

According to Jackson and Zave¹ [10], events can be classified as follows:

1. events controlled by the environment and not shared with the software system,
2. events controlled by the environment but observable by the software system,
3. events controlled by the software system and observable by the environment,

¹ They use the term “machine” for what we call software system.

4. events controlled by the software system and not shared with the environment.

If system design is not yet finished, classifying the events means deciding which part of the task to be implemented will be fulfilled by the software system. If system design is already completed, then the classification of the events must be consistent with the system design. Phenomena that are internal to the software system cannot be part of the requirements but only of the software specification.

The first five steps of our method can be carried out in any order or in parallel. These steps result in several documents:

- an entity-relationship diagram to fix the domain vocabulary, expressing both static and dynamic aspects of the system,
- a list of system operations, concerning the transformational part of the system,
- a list of events, together with their classification, concerning the reactive part of the system,
- an updated informal requirements document, where the fixed vocabulary is used and the results of the communications with the customer are taken into account.

These documents must be consistent: the vocabulary should contain exactly the notions occurring in the facts, assumptions, requirements, operations and events.

Step 6. Formalize facts, assumptions and requirements as constraints on the admissible traces of system events.

We express requirements, assumptions and facts referring to the current state of the system, events that happen, and the time an event happens:

$$S_1 \xrightarrow[t_1]{e_1} S_2 \xrightarrow[t_2]{e_2} \dots S_n \xrightarrow[t_n]{e_n} S_{n+1} \dots$$

The system is started in state S_1 . When event e_1 happens at t_1 , then the system enters state S_2 , and so forth. An element of a trace of the system thus contains a state, an event and a time. We require $t_i \leq t_j$ for $i \leq j$. Hence, concurrent events are possible.

Sometimes, it may be necessary to introduce predicates on the system state to be able to express the constraints formally. For example, an automatic teller machine can grant money to a customer only if the available amount is high enough. Expressing such a requirement makes it necessary to introduce a predicate *amount_available*. Such predicates, however, are only *declared* in the requirements elicitation phase. Their *definition* is part of the specification phase. For each predicate, the events that establish it and the events that falsify it must be given. These events must be shared with the software system.

Using constraints to express assertions on the behavior of the system has the following advantages:

- It is possible to express *negative* requirements, i.e., to require that certain things do not happen.

- It is possible to give scenarios, i.e., example behaviors of the system.
- Giving constraints does not fix the system behavior entirely. The constraints do not restrict the specification unnecessarily. Any specification that fulfills them is permitted.

In practice, requirements will often be inconsistent. We advocate to resolve conflicts at the requirements level, because no adequate specification can be derived from inconsistent requirements and because the requirements are the basis of the contract between customers and suppliers of software products. A formal representation of requirements is much more suitable to detect inconsistencies than an informal one. We have defined a heuristic approach to detect conflicting requirements [8] that cannot be presented here for reasons of space.

2.2 Case Study : An Automatic Teller Machine

We carry out the agenda of Section 2.1 for an automatic teller machine. Because the example is fairly simple, we describe the system and fix the domain vocabulary at the same time.

Step 1: Fix the domain vocabulary. A *client* has a *card* with which a *PIN* is associated. The card can be *inserted* into a *card reader*, which is part of the *automatic teller machine (ATM)*. The client will be asked to *enter the PIN*. If three times a wrong PIN is entered, the card will be *kept*. Otherwise, the client will be asked to *enter an amount*. The ATM can either *refuse* or *grant* the amount. In the latter case, the client *takes the money*, and the ATM *debits* the *account* of the client, which is *managed* by a *bank*. Before money is granted, the client can *cancel the transaction* at any time. When the transaction is terminated, the card reader *ejects* the card, and the customer will *take the card*. The bank that runs the ATM can *query* the amount available in the ATM, and it can *recharge* the ATM.

Step 2: State the facts, assumptions and requirements concerning the system. We present a selection of relevant facts, assumptions and requirements.

*fact*₁ The card reader can hold only one card at a time.

*fact*₂ If the machine is out of service, it does not read an inserted card but ejects it immediately. This is a fact because the chosen hardware behaves that way.

*ass*₁ When the machine grants money to the customer, the customer will take the money.

*ass*₂ When the machine ejects the card, the customer will take it.

- req₁* The inserted card must be valid. Otherwise, it is ejected immediately.
- req₂* A transaction can always be canceled by the customer, as long as money is not yet granted.
- req₃* To withdraw money, customers have to insert their card and enter their PIN.
- req₄* A customer has only three trials to type the right PIN. If three times a wrong PIN is entered, the card is kept by the teller machine.
- req₅* A user can only withdraw money if a weekly limit is not exceeded and if the demanded amount does not exceed the amount currently available in the teller machine.
- req₆* All valid withdrawal transactions entail a withdraw order to the bank of the client.
- req₇* The bank can recharge the machine with money between two withdrawal transactions.
- req₈* The bank can query the amount of money available in the teller machine.

Step 3: List the possible system operations that can be invoked by the users. As far as the non-reactive aspects of the system are concerned, we have two system operations that can be invoked by the bank to maintain the teller machine, one to recharge the machine with money, the other to query the amount of money available in the machine.

Step 4: List all relevant events that can happen in connection with the system, together with their parameters. For the teller machine, we can identify the following events: *insert_card*, *enter_PIN*($p : PIN$), *enter_amount*($n : \mathbb{N}$), *grant_amount*($n : \mathbb{N}$), *refuse_amount*, *debit_account*($c : CARD, n : \mathbb{N}$), *take_money*, *eject_card*, *keep_card*, *take_card*, and *cancel_transaction*.

Step 5: Classify the events. We classify the events as follows:

- Environment controlled and shared with software system are the events *insert_card*, *enter_PIN*($p : PIN$), *enter_amount*($n : \mathbb{N}$), *take_card*, *cancel_transaction*.
- We assume that the hardware of the automatic teller machine does not have a sensor to detect if the client really takes the money. Hence, the event *take_money* is environment controlled and not shared with software system.
- Controlled by software system and shared with environment are the events *grant_amount*($n : \mathbb{N}$), *refuse_amount*, *debit_account*($c : CARD, n : \mathbb{N}$), *eject_card* and *keep_card*.
- As required by the validation condition associated with this step, there are no events that are controlled by the software system but not shared with the environment.

An entity-relationship-like diagram that summarizes the results of the informal steps of the agenda is given in Figure 1. Boxes denote entities, and diamonds denote relations. Arrows indicate the direction of the relations. The relation *has* denotes static aspects of the system, whereas all other relations concern dynamic aspects.

The reader can easily check the validation condition associated with Steps 1–4: the diagram contains exactly the notions used to express the facts, assumptions, requirements, operations and events.

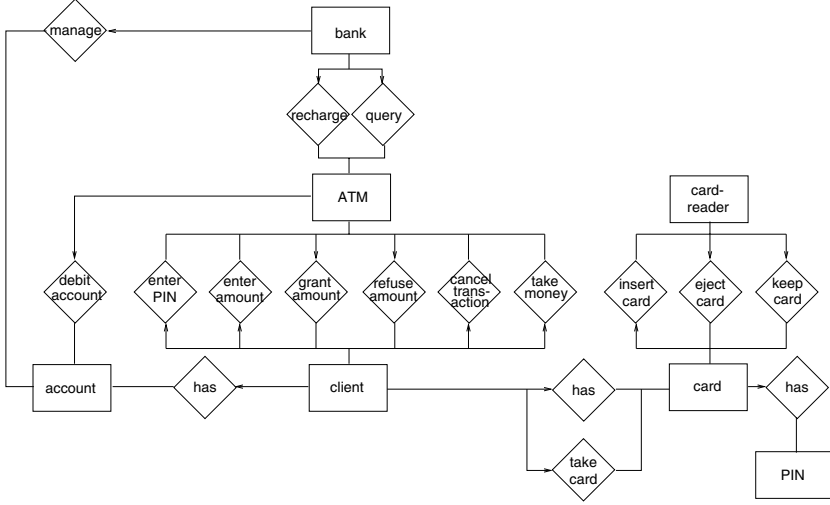


Fig. 1. Domain vocabulary for the teller machine

Step 6: Formalize facts, assumptions and requirements as constraints on the admissible traces of system events. For each system, we call the set of possible traces Tr . Constraints are expressed as formulas restricting the set Tr . For a trace $tr \in Tr$, $tr(i)$ denotes the i -th trace element, $tr(i).s$ denotes the i -th state that is reached, and $tr(i).e$ denotes the event that happens in that state. For each possible trace, its prefixes are also possible traces. The definitions of the specification macros *immediately_followed_by* and *subtraces* used in the following are obvious.

For reasons of space, we only present the formalizations of $fact_1$, ass_1 , req_1 and req_4 .

$$\underline{fact_1} \forall tr : Tr \bullet (\forall i : \text{dom } tr \bullet \text{card\_inside}(tr(i).s) \Rightarrow tr(i).e \neq \text{insert\_card})$$

To formalize $fact_1$, we have introduced a predicate *card_inside* on the system state. The predicate is established by the event *insert_card*, and it is falsified by the events *eject_card* and *keep_card*.

$$\underline{ass_1} \forall n : \mathbb{N}_1 \bullet \text{grant\_amount}(n) \text{ immediately\_followed\_by } \text{take\_money}$$

$$\underline{req_1} \forall tr : Tr \bullet (\forall i : \text{dom } tr \mid i \neq \#tr \bullet \\ (tr(i).e = \text{insert\_card} \wedge tr(i+1).e \neq \text{eject\_card}) \\ \Rightarrow \text{valid\_card}(tr(i+1).s)))$$

To formalize req_1 , we have introduced a predicate *valid_card* on the system state. If an inserted card is not ejected immediately, the state reached after insertion of the card must satisfy the predicate *valid_card*. The predicate *valid_card*

is established by the event *insert_card* and falsified by the events *eject_card* and *keep_card*.

$$\begin{aligned}
 & \underline{req_4} \quad \forall tr : Tr \bullet \\
 & \quad (\forall tr' : \text{subtraces}(tr, \text{insert\_card}, \text{grant\_amount}) \bullet \\
 & \quad \quad \#\{trit : \text{ran } tr' \mid \exists p : PIN \bullet trit.e = \text{enter\_PIN}(p)\} \leq 3 \wedge \\
 & \quad \quad \exists i : \text{dom } tr'; p : PIN \mid ti.e = \text{enter\_PIN}(p) \bullet \\
 & \quad \quad \quad \text{right\_PIN}(tr'(i+1).s)) \\
 & \quad \wedge \quad (\forall tr' : \text{subtraces}(tr, \text{insert\_card}, \text{insert\_card}) \mid \\
 & \quad \quad (\#\{trit : \text{ran } tr' \mid \exists p : PIN \bullet trit.e = \text{enter\_PIN}(p)\} = 3 \wedge \\
 & \quad \quad (\forall i : \text{dom } tr' \mid \exists p : PIN \bullet tr'(i).e = \text{enter\_PIN}(p) \bullet \\
 & \quad \quad \quad \text{wrong\_PIN}(tr'(i+1).s))) \bullet \\
 & \quad \quad (\exists j : \text{dom } tr' \bullet tr'(j).e = \text{keep\_card}))
 \end{aligned}$$

The first conjunct of the formula states that if money is granted, then at most three times a PIN has been entered, and one of them must have been valid². The second conjunct of the formula states that if three times the wrong PIN is entered, then a *keep_card* event occurs before the next *insert_card* event. To express this constraint, have introduced predicates *right_PIN* and *wrong_PIN* on states that express whether an entered PIN is valid or not.

Not only the formal constraints, but also the documents produced during the first five steps of the agenda form the starting point for the specification phase.

3 Specification Development

Jackson and Zave [10] consider a specification to be a special kind of requirement. A requirement is a specification if all events constrained by the requirement are controlled by the software system, and all information it relies on is shared with the software system and refers only to the past, not the future. Requirements (and thus specifications) do not make statements about the state of the software system. In contrast to this view, we consider a specification to be a *model* of the software system to be built in order to satisfy the requirements. It forms the basis for refinement and implementation. Therefore, in our approach, a specification may - in contrast to the requirements - make statements about the software system that are not directly observable by the environment.

While requirements elicitation is independent of the specification language that is used, the development of a specification depends to a certain extent on the specification language and its means of expression. In the following, we use the specification language Z [16]. If other specification languages are used, our method is applicable, too. In this case, slight changes of the agenda might be necessary.

² Note that we choose not to prescribe that after a correct PIN has been entered, no more *enter_PIN* events will occur. This will be decided in the specification.

3.1 Agenda for Specification Development from Requirements

The starting point of the specification development is the whole material obtained by the requirements elicitation phase presented in Section 2. Again, our method for specification acquisition is expressed as an agenda, given in Table 2. The steps have to be performed in the given order.

No	Step	Validation Conditions
1	Define a first approximation of the software system state and the initial states.	
2	Augment the specification, incorporating the requirements one by one. For each constraint:	⊢ The constraints expressing facts must not be violated. ○ All events introduced in the requirements elicitation phase must be taken into account. ○ The preconditions of the operations must be defined appropriately. ⊢ The post-states of the operations must be completely defined. ⊢ Each predicate on states used in the requirements must be definable with the final definition of the system state.
	2.1 List the events occurring in the constraint.	
	2.2 For each event in the list, set up a first definition of the corresponding Z operation, or adjust an already existing operation.	

Table 2. Agenda for specification acquisition

Step 1. Define a first approximation of the software system state.

This approximation should be defined in such a way that as many as possible of the predicates on the system state that were introduced in the requirements elicitation process can be defined. The initial states are specified in parallel with the legal states.

Step 2. Augment the specification, incorporating the requirements one by one.

For each constraint, this step should be performed in two sub-steps. The basic idea is to define a Z operation for each event identified in Step 4 of the requirements elicitation phase that is shared with the software system (according to the classification made in Step 5), and for each system operation identified in Step 3.

Step 2.1 can be performed by a simple syntactic inspection of the constraint in question. Step 2.2, however, can be complex with several revisions of the current version of the specification [12]. The state of the system and the operations have to be re-considered in order to take into account the evolution introduced by new constraints. Propagation of modifications is important here [15]. Incorporating a new constraint may involve the following modifications: (i) adding or modifying state components, (ii) adding or modifying data types, (iii) adding or modifying the state invariant and (iv) propagating those modifications into the current state of the specification.

The first two validation conditions require that all definitions and modifications must respect the domain properties as they are expressed by the facts, and that no event may be “forgotten”.

To explain the third validation condition, we note that the preconditions for system operations (see Step 3 of the agenda for requirements elicitation) should be as weak as possible to make the software system more robust. For operations corresponding to the reactive part of the system, however, the situation is different. Strong preconditions may enforce a certain order in which operations can be applied. This is a possible way to encode behavioral descriptions in Z. In our case study (see Section 3.2), we will make use of this possibility.

Fourth, in most cases, an operation should give a condition for all state components that specifies their value after the operation has terminated. If non-determinism is introduced deliberately, this should be justified.

The last validation condition refers to the final definition of the system state. It must contain enough information to define the predicates introduced to express constraints in the requirements elicitation phase.

The agendas for requirements elicitation and specification acquisition provide an integrated approach that introduces formality as early as possible in the software engineering process. The formal expression of requirements and facts as constraints guide the development of the formal specification.

Our approach even allows to define a *notion of correctness* of a specification with respect to requirements, facts and assumptions: the set of possible traces of the specification is the set of traces where each operation is executed only if its precondition is satisfied. Assuming that the assumption constraints are satisfied, it must be demonstrated that the set of possible traces induced by the specification fulfills the constraints stated as requirements and facts.

3.2 Case Study : The Automatic Teller Machine

Taking the result of the requirements elicitation phase of Section 2.2 as the starting point, we now use the agenda presented in Section 3.1 to develop a formal Z specification of the automatic teller machine.

Step 1: Define a first approximation of the software system state.

In formalizing the requirements, we have introduced the predicates *card_inside_out_of_service*(*st* : *STATE*), *valid_card*(*st* : *STATE*), *right_PIN*(*st* : *STATE*), *wrong_PIN*(*st* : *STATE*), *amount_available*(*st* : *STATE*, *n* : $\mathbb{N}$) and *customer_limit_not_exceeded*(*st* : *STATE*, *c* : *CARD*, *n* : $\mathbb{N}$). These motivate the following definitions:

With this preliminary definition of the system state, we cannot define the predicates *right_PIN*(*st* : *STATE*) and *wrong_PIN*(*st* : *STATE*). However, we decide that the PIN should only be an input, and not a state component. We have chosen to introduce a state component that stores the introduced card, because we will need more information about this card in the following. The predicate *card_inside* can be defined as *card* $\neq$ *null_card*.

$OP\_MODES ::= out\_of\_service \mid in\_service$ $[CARD, PIN]$ $\frac{}{valid\_cards : \mathbb{P} CARD}$ $\frac{}{null\_card : CARD}$ $\frac{}{null\_card \notin valid\_cards}$	$\frac{}{get\_PIN : CARD \leftrightarrow PIN}$ $\frac{}{possible\_withdraw : CARD \leftrightarrow \mathbb{N}}$ $\frac{}{valid\_cards \subseteq \text{dom } get\_PIN}$ $\frac{}{valid\_cards \subseteq \text{dom } possible\_withdraw}$
$\frac{}{Teller\_machine}$ $current\_card : CARD$ $op\_mode : OP\_MODES$ $available\_amount : \mathbb{N}$ $(current\_card = null\_card \Leftrightarrow op\_mode \in \{in\_service, out\_of\_service\})$ $(current\_card \in valid\_cards \Leftrightarrow op\_mode \notin \{in\_service, out\_of\_service\})$	$\frac{}{Init\_Teller\_machine}$ $Teller\_machine'$ $given? : \mathbb{N}$ $op\_mode' = in\_service$ $current\_card' = null\_card$ $available\_amount' = given?$

Step 2: Augment the specification, incorporating the requirements one by one. For each event which is shared with the software system, we introduce an operation. The definition of each operation is guided by analyzing each requirement constraint, taking also into account the domain properties expressed by the facts.

Goal: req_1 The inserted card must be valid. Otherwise, it is ejected immediately.

List of events: $insert\_card$, $eject\_card$

To define first approximation of the $insert\_card$ operation derived from req_1 , we introduce a new value $valid\_card$ for the operational modes of the teller machine, which must be added to the type OP_MODES . We also must define a second operation, $eject\_card$, that returns the card to the customer:

$\frac{}{insert\_card}$ $\Delta Teller\_machine$ $card? : CARD$ $op\_mode \in \{in\_service, out\_of\_service\}$ $(op\_mode = out\_of\_service \vee card? \notin valid\_cards) \Rightarrow eject\_card$ $(op\_mode = in\_service \wedge card? \in valid\_cards \Rightarrow current\_card' = card? \wedge op\_mode' = valid\_card \wedge available\_amount' = available\_amount)$	$\frac{}{eject\_card}$ $\Delta Teller\_machine$ $current\_card' = null\_card$ $op\_mode \neq out\_of\_service \Rightarrow op\_mode' = in\_service$ $op\_mode = out\_of\_service \Rightarrow op\_mode' = op\_mode$ $available\_amount' = available\_amount$

Note that $card \neq null\_card$ is not a precondition for the operation $eject\_card$, because $fact_2$ states that an invalid card is ejected immediately, as defined in $insert\_card$. To take $fact_1$ into account, we have given $insert\_card$ the precondition $op\_mode \in \{in\_service, out\_of\_service\}$. In the following, we must guaran-

tee that this precondition is established only by the operations *eject_card* and *keep_card*.

Goal: *req₄* A customer has only three trials to type the right PIN. If three times a wrong PIN is entered, the card is kept by the teller machine.

List of events: *insert_card*, *grant_amount*, *enter_PIN(p)*, *keep_card*

We have to add new values to the *op_mode* state component to count the number of trials of the customer and to express if the customer is granted the money or the card is kept.

OP_MODES::=*out_of_service* | *in_service* | *valid_card* | *PIN_entered*
| *incorrect_PIN1* | *incorrect_PIN2* | *failure* | *success*

— <i>enter_PIN</i> — Δ<i>Teller_machine</i> <i>pin?</i> : <i>PIN</i> <i>op_mode</i> $\in$ {<i>valid_card</i>, <i>incorrect_PIN1</i>, <i>incorrect_PIN2</i>} <i>get_PIN</i>(<i>current_card</i>) = <i>pin?</i> $\Rightarrow$ <i>op_mode'</i> = <i>PIN_entered</i> <i>get_PIN</i>(<i>current_card</i>) $\neq$ <i>pin?</i> $\Rightarrow$ (<i>op_mode</i> = <i>valid_card</i> $\Rightarrow$ <i>op_mode'</i> = <i>incorrect_PIN1</i>) $\wedge$ (<i>op_mode</i> = <i>incorrect_PIN1</i> $\Rightarrow$ <i>op_mode'</i> = <i>incorrect_PIN2</i>) $\wedge$ (<i>op_mode</i> = <i>incorrect_PIN2</i> $\Rightarrow$ <i>op_mode'</i> = <i>failure</i>) <i>current_card'</i> = <i>current_card</i> <i>available_amount'</i> = <i>available_amount</i>

— <i>keep_card</i> — Δ<i>Teller_machine</i> <i>op_mode</i> = <i>failure</i> <i>current_card'</i> = <i>null_card</i> <i>op_mode'</i> = <i>in_service</i> <i>kept_cards'</i> = <i>kept_cards</i> $\cup$ {<i>current_card</i>} <i>available_amount'</i> = <i>available_amount</i>	— <i>grant_amount</i> — Δ<i>Teller_machine</i> <i>op_mode</i> = <i>PIN_entered</i> <i>op_mode'</i> = <i>success</i>

With these definitions, *fact₁* is not violated, because only *keep_card* establishes the precondition of *insert_card*. The operation *grant_amount* will be defined further when taking into account *req₅* and *req₆*.

A new state component *kept_cards* : $\mathbb{P}$ *CARD* must be added to the schema *Teller_machine*, and the current specification has to be revised by adding the equation *kept_cards'* = *kept_cards* to each operation schema except *keep_card*.

Note that in the requirements we have no statement about what happens with the kept cards. As it is now, the set *kept_cards* can only be augmented. Thus, we have detected a missing requirement.

The validation conditions of the agenda are fulfilled, as can be checked by inspection of the specification. As an application-dependent validation of the teller

machine specification, we can check if all operational modes that were introduced during the specification acquisition phase are indeed reachable from an initial state. It turns out that we have introduced a mode, *out_of_service*, without any possibility to access it. Again, we have detected a missing requirement, which should cause us to add a requirement describing when the teller machine goes out of service.

4 Related Work

Separating domain knowledge from requirements and checking consistency between these is frequent in the literature. For example, when Parnas describes the mathematical content of a requirements document for a nuclear shutdown system [14], he introduces different mathematical relations, one describing the environment of the computer system, and one describing the requirements of the computer system. The two relations must fulfill certain feasibility conditions. Parnas also notes that a critical step in documenting the requirements concerns the identification of the environmental quantities to be measured or controlled and the representation of these quantities by mathematical variables. He proposes to characterize environmental quantities as either monitored or controlled. This corresponds to the classification of events in our and Jackson and Zave's approach [10]. Reading a monitored quantity corresponds to an event observable by the software system, and controlling a quantity corresponds to events controlled by the software system.

Whereas assumptions are not used by Parnas and by Jackson and Zave, they do play a role in KAOS [2,3]. KAOS supports the design of composite systems (see also [5]). Such a system consists of human, software and hardware agents, each of them being assigned responsibility for some goals. The KAOS approach is goal-oriented: goals are stated and then elaborated into KAOS specifications in several consecutive steps, starting with elaborating the goals in an AND/OR structure and ending with the assignment of responsibilities to agents. KAOS is similar to our approach in that it provides heuristics for requirements elicitation and specification development. It is distinguished from our approach in that it uses its own language and in that it takes a much broader perspective: not only the software system, but also its environment are modeled in detail. This results in a very rich terminology. In contrast, our approach is focussed on developing a formal specification for a software component. We only model those aspects of the environment that are necessary for an adequate specification.

Easterbrook and Nuseibeh [4] do not distinguish different phases for requirements elicitation and specification development. They elicit more requirements when they detect inconsistencies in their specifications. On the one hand, their approach makes it possible to delay the resolution of conflicts. On the other hand, this kind of "lazy" requirements elicitation delays the point where a definite contract between customers and providers of the software product can be made. Moreover, it is harder to validate a specification with respect to the requirements when no separate requirements document is set up that is checked for consistency and completeness in its own right.

We have already contrasted our approach to the one of Jackson and Zave [10,17]: first, we see a difference in requirements and specifications. As a re-

sult, our method leads to a formal specification that is expressed in a conventional specification language, whereas their approach stops when the requirements have been transformed in such a way that they can be regarded as a specification. The language in which the requirements are expressed (formulas of first-order predicate logic) is not changed during this process. Second, the most important part of our approach is the methodological guidance that is given to analysts and specifiers in form of agendas, as well as the validation of the developed products. These issues are not addressed explicitly by Jackson and Zave.

Of the object-oriented methods, we have only borrowed the very first steps, where the relevant vocabulary is introduced in a systematic way. Our approach is not biased toward object-oriented development.

5 Conclusions

Requirements define a set of conditions that must be met by a system or a system component to satisfy a contract, standard or other imposed document or description. For example, the IEEE Standard 1498 [9] defines a requirement as a characteristic that a system or a software item must possess in order to be acceptable to the acquirer. Requirements should be completely and unambiguously stated. In our approach, achieving completeness is supported by the first steps of the requirements elicitation agenda, where brainstorming processes are performed, and by feedback from the specification phase. In our example, we found missing requirements by analyzing the formal specification. Formalizing the requirements as we do eliminates ambiguities.

Adequately integrating formal methods into the whole development process is still an important challenge in software engineering. Integration strategies have been classified by Fraser et al. [6] with respect to the following factors:

1. Does the strategy lead directly from the informal requirements to the formalized specification, or does it introduce intermediate and increasingly formal models of the requirements?
2. If the strategy introduces intermediate models, is the process one of parallel, successive refinement of the requirements and the formal specification, or are the formal specifications derived after the requirements models have been finalized in a sequential strategy?
3. To what extent does the strategy offer mechanized support for requirements capture and formalization?

In terms of these classification criteria, our approach can be classified as *transitional*, because we introduce predicates over system event traces as an intermediate representation of the requirements. Our approach is *sequential*, because the specification phase is only entered when the requirements are deemed complete and correct. However, as already mentioned, feedback between the two phases is possible. Mechanized support for our approach is not yet available. Such support is conceivable for checking validation conditions, performing conflict analyses, and for developing the specification. First steps in this direction have already been performed: there is a prototypical implementation of our algorithm to detect conflicting requirements [8], and, furthermore, with Proplane [13]

a specification support system already exists that can be adjusted to support the method presented here.

Requirements traceability is an important issue in requirements engineering. Jarke [11] defines requirements traceability as the ability to describe and follow the life of a requirement, in both a forward and backward direction. In our method, traceability is guaranteed in the following way:

- Single requirements are fragments as small as possible. The smaller the requirements, the better traceable they are, because their realization does not distribute over large parts of the system.
- For each event and each predicate that is introduced, it is noted in which requirements it is used.
- For each part of the formal specification, we can name the requirements that lead us to define it in the way we did.

In summary, our approach can be characterized as follows: requirements/requirements elicitation on the one hand and specifications/specification acquisition on the other hand are clearly distinguished. We give substantial methodological guidance for the two activities, which are integrated smoothly. Our approach does not introduce a new language or a new formalism. The requirements elicitation phase is independent of the specification language to be used, and the specification development phase can be adjusted to support the usage of other specification languages than Z. The method is suitable for transformational as well as reactive systems. Also real-time considerations can be taken into account.

We propose a standardized way of expressing facts, assumptions and requirements. Constraints on the set of possible traces are a very flexible and powerful means of describing a system and its interaction with the environment. Expressing requirements as constraints on traces makes it possible to systematically detect conflicting requirements and to define a formal notion of correctness of a specification with respect to a set of requirements.

References

1. P. Chen. The entity-relationship model – towards a unified view of data. *ACM Transactions on Database Systems*, 1(1), 1976. 311
2. A. Dardenne, A.v. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993. 322
3. R. Darimont and A.v. Lamsweerde. Formal Refinement for Patterns for Goal-Driven Requirements Elaboration. In *Proc FSE-4, ACM Symposium on the Foundation of Software Engineering*, pages 179–190, 1996. 322
4. S. Easterbrook and B. Nuseibeh. Using ViewPoints for inconsistency management. *Software Engineering Journal*, pages 31–43, January 1996. 322
5. M.S. Feather, S. Fickas, and R.B. Helm. Composite System Design : the Good News and the Bad News. In *Proc. 6th Knowledge-Based Software Engineering Conference*, pages 16–25. IEEE Computer Society Press, 1991. 322
6. M.D. Fraser, K. Kumar, and V.K. Vaisnavi. Strategies for Incorporating Formal Specifications in Software Development. *CACM*, 37(10):74–86, Oct. 1994. 323
7. M. Heisel. Agendas – a concept to guide software development activities. In R. N. Horspool, editor, *Proc. Systems Implementation 2000*, pages 19–32. Chapman & Hall, 1998. 310

8. M. Heisel and J. Souquière. A heuristic approach to detect feature interactions in requirements. In K. Kimbler and W. Bouma, editors, *Proc. 5th Feature Interaction Workshop*, pages 165–171. IOS Press Amsterdam, 1998. 314, 323
9. IEEE94. Software development. IEEE publications office, IEEE Standard 1498, Los Alamitos, CA, March 1994. 323
10. M. Jackson and P. Zave. Deriving specifications from requirements: an example. In *Proceedings 17th Int. Conf. on Software Engineering, Seattle, USA*, pages 15–24. ACM Press, 1995. 310, 312, 317, 322, 322
11. M. Jarke. Requirements tracing. *Communications of the ACM*, pages 32–36, December 1998. 324
12. N. Lévy and J. Souquière. A “Coming and Going” Approach to Scenario. In W. Schafer, J. Kramer, and A. Wolf, editors, *Proc. 8th Int. Workshop on Software Specification and Design*, pages 115–158. IEEE Computer Society Press, 1996. 318
13. N. Lévy and J. Souquière. Modelling Specification Construction by Successive Approximations. In M. Johnson, editor, *6th International AMAST conference*, pages 351–364. Springer Verlag LNCS 1349, 1997. 323
14. D.L. Parnas. Using Mathematical Models in the Inspection of Critical Systems. In M. Hinchey and J. Bowen, editors, *Applications of Formal Methods*, pages 17–31. Prentice Hall, 1995. 322
15. S. Sadaoui and J. Souquière. Quelques approches de la réutilisation dans le modèle Proplane. In *Conférence AFADL, Approches formelles dans l’assistance au développement de logiciels*, pages 85–96, Toulouse, 1997. Onera–Cert. 318
16. J. M. Spivey. *The Z Notation – A Reference Manual*. Prentice Hall, 2nd edition, 1992. 317
17. P. Zave and M. Jackson. Four dark corners for requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, January 1997. 310, 322

Dealing with Semantic Heterogeneity during Data Integration

Zoubida Kedad¹ and Elisabeth Métais¹

¹ Laboratoire PRiSM, Université de Versailles, 45 avenue des Etats-Unis
F-78035 Versailles cedex, France
{Zoubida.Kedad, Elisabeth.Metais}@prism.uvsq.fr

Abstract. Multi-sources information systems, such as data warehouse systems, involve heterogeneous sources. In this paper, we deal with the semantic heterogeneity of the data instances. Problems may occur when confronting sources, each time different level of denominations have been used for the same value, e.g. "vermilion" in one source, and "red" in an other. We propose to manage this semantic heterogeneity by using a linguistic dictionary. "Semantic operators" allow a linguistic flexibility in the queries, e.g. two tuples with the values "red" and "vermilion" could match in a semantic join on the "color" attribute. A particularity of our approach is it states the scope of the flexibility by defining classes of equivalent values by the mean of "priority nodes". They are used as parameters for allowing the user to define the scope of the flexibility in a very natural manner, without specifying any distance.

1 Introduction

A Multi-Source Information System (MSIS) is composed of a set of heterogeneous and distributed data sources, and a set of views (or queries) defining user requirements over these sources. Examples of MSIS are web systems and data warehouse systems. The main difference between an MSIS and a classical information system resides in its definition and its feeding with data. While the database schema of a classical information system is defined as an integrated data structure, the database structure of an MSIS is defined as a set of (possibly independent) views. Moreover, while the feeding of the first database is done by the users through their applications, the feeding of the second database is automatically done by the system from the data sources. MSIS applications aim only at the use of this data without any direct update.

One possible solution to access distributed and heterogeneous data sources is to build on top of the local systems a global system which integrates the different data source [23],[27]. The integration, exportation and importation of data is done by the federated system. Several systems represent this architecture such as the TSIMMIS system [4],[18], Information Manifold [17], and SIMS [2].

An alternative to provide access to a set of distributed and heterogeneous data sources is the data warehouse (DW) approach, in which the relevant data is extracted from each source, then translated, cleaned and integrated to be stored in a centralized

repository. Among the systems representing this approach, we can find the WHIPS prototype [10], H2O [28] and DWQ [11].

MSIS have introduced new design activities such as the selection of relevant data sources, the query containment problem [17], the selection of views to materialize [26], the update propagation of source changes [15], the definition and generation of mediators [27], the definition of computing expressions for each view in the system [14] and the data integration and cleaning [5],[1],[3].

One key problem in designing MSIS is the reconciliation of the data contained in the data sources. When merging the values contained in the sources, the following problems may occur :

- Differences in data format, for example when the instances of two equivalent attributes have different lengths;
- Differences in scale, when the instances of two equivalent attributes are expressed using different units, such as French francs and dollars;
- Differences in the encoding of attributes, for example the attributes *sex* in two different data sources can be encoded using the set of instances {1, 2} and {F, M} respectively;
- Differences in the denominations of the instances, for example, the same attribute *color* may have the two values "azure" and "blue".

Natural language techniques could be useful for solving some of the cases in which the denominations of the instances are different. These techniques have been used for schema integration. [8] proposes the use of fuzzy and incomplete terminological knowledge to determine the correspondence assertions between the compared objects. In [12] and [13], correspondence assertions between the schemas to integrate are determined using case grammar and speech act theory. [19] and [20] suggest the use of semantic dictionaries storing concept hierarchies and canonical graphs to retrieve the semantics of the schemas. [21] uses a fuzzy thesaurus built using linguistic tools to compute a fuzzy semantic nearness coefficient between objects. In the Carnot project [24], which provides a framework for heterogeneous database integration, a common ontology is used to semantically relate different schemas with each other and thereby enable the interoperation of the underlying databases. The common ontology is expressed using CYC, a knowledge base providing the "commonsense" knowledge of an application domain.

In this paper we propose to deal with the semantic heterogeneity, by using meta data including a linguistic dictionary. The main idea of this technique is to provide linguistic flexibility in the queries, without any distance set by the user. Section 2 gives an overview of our objective. Section 3 discusses the linguistic techniques supporting the methodology and sections 4 and 5 detail the proposed semantic operators.

2. The General Approach

Using traditional relational algebra operators for querying a MSIS may lead to meaningless results; the example of figure 1 enlightens the problem we aim to solve. We can see on this example that the result of this union is hard to interpret and that it contains inconsistencies. Flexible relations as defined in [1] allow a correct chaining with other operators, without propagating errors. However, our purpose is different in the sense that we want to obtain the result of Fig. 2, by the way of "semantic operators" which are widely described in section 4.

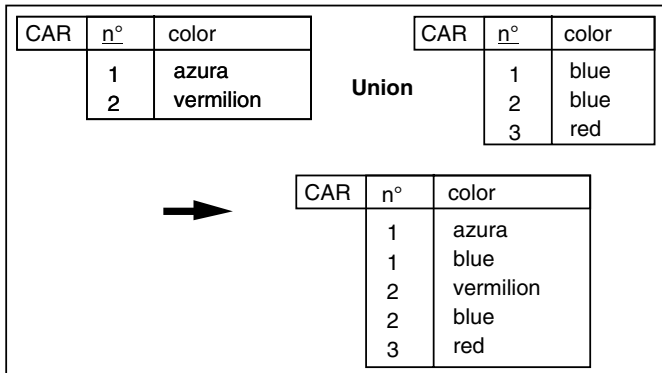


Fig. 1. : Union without linguistic knowledge

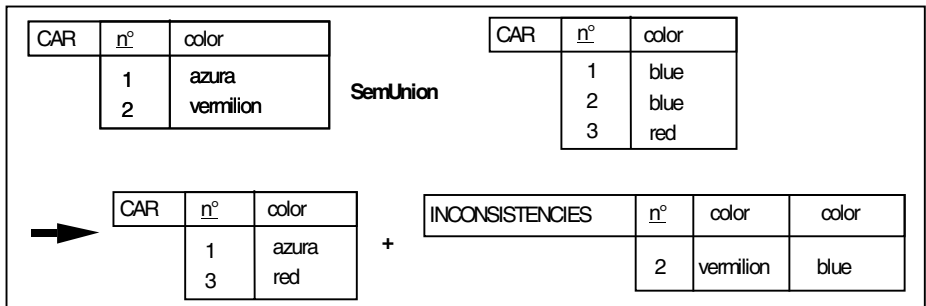


Fig. 2. : Semantic Union using Linguistic knowledge

Remark : In Fig. 2 and in the remaining of this paper, we use the term "inconsistency" for the case of two tuples "matching" on all the key attribute values but having conflicting values for some non-key attributes, as it is used in [6] and [1].

The basic problem is that, giving the two source schemas $R1(K, A1, A2, A3)$ and $R2(K', A4, A5, A6)$, once the correspondences between attribute names are stated at the schema level (for example $A1$ is equivalent to $A4$), it is necessary to have the knowledge on the instance level (the $a1$ value for a $r1$ tuple of $R1$ is the same as the $a4$ value of a $r2$ tuple in $R2$, in spite of their coding heterogeneity).

Considering the thousands of tuples from the tens sources, it is not realistic to store these assertions, like schema assertions are. Furthermore, these assertions are partially user dependant. For example, for the same query "search for cars having the same color than their cover-seats", some users expect blue cars and covers, whatever the kind of blue are, while other users expect only cars and cover both light-blue, or cars and covers both deep-blue.

A preliminary cleansing of the data is also inadequate because the denominations are not exactly equivalent (on the opposite of a code "1" and a code "M" for "male"). Thus, cleaning in the first relation "azure" in "blue" will lead to a lack of semantic; and cleaning in the second relation "blue" in "azure" is impossible before having the knowledge that the value in the other relation is "azure" and not "navy".

Consequently, our approach integrates the linguistic matching in the processing of the queries. These queries may have different purposes and may come from different "users" (e.g. user, interfaces, mediators).

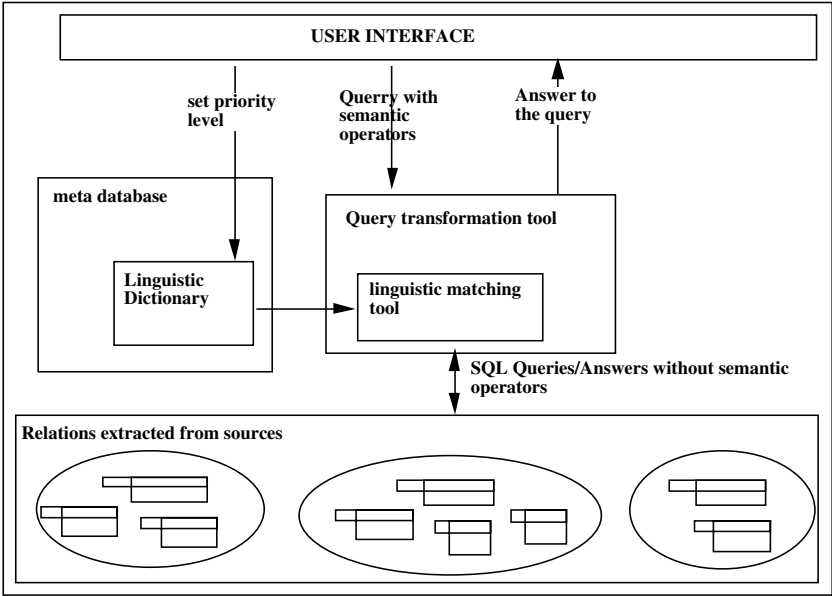


Fig. 3. : Global methodology

A linguistic matching (i.e. a linguistic flexibility in comparison of values) is proposed to the user by the mean of a set of **semantic operators**, as the one of Fig. 2. These operators are namely the SemUnion, the SemJoin, the SemRestriction, the SemDifference and the SemIntersection. It is important to notice that the linguistic flexibility must be triggered by the user since one user may search the red cars, and expect vermillion cars and other kind of red cars, while an other user may search for red cars, expecting cars having exactly the "red" value for the attribute "color". This is a important difference with other cleaning procedures such as converting moneys,

which have no impact on the semantic of the query, and thus may be automatically performed.

A traditional challenge when introducing some flexibility is to define its scope. Numerous works [22], [25] have proposed semantic distances for integration or retrieval purposes. However, although this distance could be computed using a dictionary, we didn't want the user to specify it because this is not a natural way of thinking. In our approach, the user automatically defines classes of values, by the mean of **priority nodes** in a hierarchical dictionary. Then, two values are considered semantically close if they belong to the same class, and semantically far if they do not.

The semantic operators are transformed into classical operators and then executed by the DBMS on the relations extracted from the sources as shown in figure 3. This transformation need metadata which are provided by a dictionary adapted from linguistic dictionaries.

3 The Linguistic Knowledge Supplied by the Meta Database

Besides procedures and table conversions, the meta database has to contain a linguistic dictionary in order to reconcile denominations. In this section, we will describe how linguistic dictionaries may be customized in order to be more suitable for the purpose of data reconciliation in MSIS.

3.1 General Linguistic Dictionaries

Several projects aiming at building electronic linguistic dictionaries are in progress such as CYC [16], EDR [7] and WordNet [9]. Beyond morphologic and syntactic information they provide semantic links between words such as synonymy, links relating words to concepts and links between concepts such as antonymy, hyponymy/hypernymy (is-a) and meronymy/ holonymy (part-of). The is-a links are useful for conceptual schema elaboration and are organized in a "hierarchy of concepts". Other links are supported by "canonical graphs" which specify relationships expected among the concepts involved in a given action.

These general linguistic dictionaries are very helpful for retrieving the semantic of the nouns in schemas or data for their integration. They allow to supply with all the semantic lost during the modeling process. In previous works [19] [20] we used WordNet to integrate database schemas. For example, one contribution of such dictionaries is the detection of generalization links between objects of different views. Another contribution is the deduction of the roles played by each entity in a relationship.

General linguistic dictionaries are very useful for schema integration. However, they seem to have some limitations for data integration in a MSIS, mainly at three levels: the vocabulary, the links and the normalization.

Their vocabularies are adapted to describe most common concepts of the real world, and then are very suitable for knowledge representation, including databases schemas. But they can not catch all the possible values of instances. Most of them are specific to their owner, and only part of them may be considered as word of the common vocabulary.

The links they provide are also suitable for the schema level (semantic roles, converse verbs,...), but other links would be closer to the data's problems. For example, the cardinality, from an object to a property should help in decision taking, when the same property has different values in distinct sources. Such a link is not found in a general dictionary, but is one of the basic links between data.

Concerning normalisation, general dictionaries do not provide a preference among a set of synonyms, or inside a branch of the concept hierarchy.

3.2 Proposal for a More Suitable Linguistic Dictionary

Due to the limitations described above we propose to enrich the linguistic dictionary so that it is more data reconciliation oriented. We will illustrate this section with the example of a dictionary for the color denominations, mainly devoted to the vehicle's colors. Colors are essential in car marketing and are mined for many selling analysis. This example illustrates the case of data having data which may have a huge number of denominations since they are usually specific to a constructor.

The Concept Hierarchy

The main structure of the linguistic dictionary is the **concept hierarchy**, which stores the "is-a" relationship. As we can see Fig. 4, the leaves nodes of the concept hierarchy are often enterprise's denominations. This example enlightens a particularity of linguistic dictionary devoted to data instances, which is to gather both common vocabulary names and names that looks like codes, integrated in the same hierarchy.

The elaboration of such a dictionary requires the cooperation of the main concerned enterprises. Thus an automatic procedure has to be proposed in order to feed the dictionary with data. Concerning the particular example of the color denominations, this procedure does exist. A color may be encoded with three coordinates according to the three basic colors axis. While some colors have only one point in this space (usually constructor colors such as "Glacier"), others (usually common name colors such as "red") cover a volume in this space. Then the hierarchy is automatically build as follow: a color **subsumes** another color if the corresponding volume contains the volume of the other color (which may be a point).

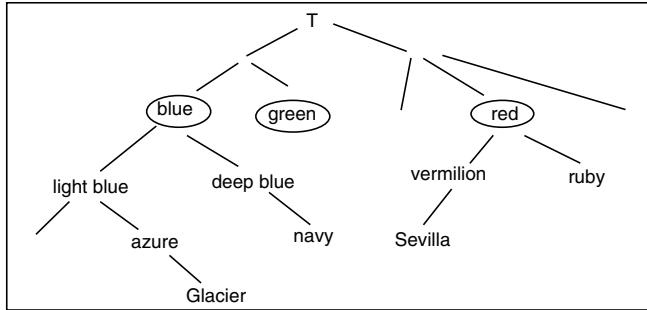


Fig. 4. : Excerpt of the concept hierarchy

We notice that as other cleaning metadata, a part of the dictionary may be shared by a large number of applications and users, and an other part is an enrichment which is specific to an application. The dictionary also contains a set of parameters which may be updated for each user. They are described in the following section.

The Priority Level

We can intuitively introduce the notion of **priority nodes** in the hierarchy as a way for the user to specify classes of equivalent values. These nodes are graphically represented by a cercle as shown Fig. 4. where "blue", "green" and "red" are priority nodes. **The class induced by a priority node is constituted by the sub-tree having the priority node as top** (i.e. the class of a priority node contains the set of values that it subsumes).

The choice of the priority nodes is arbitrary and may change from one query to another. The user may change the priority nodes as parameters. That is a very easy and natural manner for the user to precise his classes of semantically close values, without manipulating any distance.

However, it is important to notice that in practice, this choice is quite stable. Usually, the priority node of a branch is the most frequently referred node in the language among all the nodes of this branch. Default priority nodes could be those which are the most frequently referred in the definitions of both their subsuming and subsumed nodes. As an example, in usual dictionaries, both "vermilion" and "warm color" are defined using the "red" denomination.

The Cardinality Links

To reconcile data according to one criteria we need to know if it is possible for an object to have several values for this criteria. The dictionary store the "cardinality" relationship, well known in the database area. In our example, a table stores the cardinality from objects toward the color property. An excerpt of this table is (car, monovalued), (flag, multivalued).

3.3 Assertions Provided by a Priority Level Linguistic Dictionary

Thanks to the linguistic hierarchy and its priority level described above, the dictionary provides assertions, in order to describe the existing relation between two values. We introduce different kinds of assertions, describing respectively the equivalence relation, the inclusion relation, the co-class and the dis-class relations. These assertions will be used by the semantic operators described in the sections below.

Definition 1: Two values A and B are **equivalent** if they are either equal or synonyms.

Definition 2 : An A value **is included** in a B value if and only if there exists a path from A towards B in the concept hierarchy (in other words, if B subsumes A). Respectively, we say that A **includes** B. As an example, in the dictionary of Fig. 4, "Glacier" is included in "light blue".

Definition 3: Two values A and B are **co-class** if and only if there exists a priority node C including both A and B. For example, in the dictionary of Fig. 4, "Azure" and "Navy" are co-class, since they are both "blue" which has been stated as priority node.

Definition 4 : Two values A and B are **dis-class** if and only if there is no priority node including both A and B (that is, if they are not co-class).

4 The Semantic Operators

In this section, we introduce the semantic operators allowing to deal with linguistic heterogeneity. These operators are denoted SemUnion, SemJoin, SemRestriction, SemDifference and SemIntersection. A particular attention will be paid on the way they use the metadata provided by the linguistic dictionary.

4.1 Notation of the Semantic Operators

The semantic operators have to be parameterized with the set of attributes the linguistic matching must be applied on. For example the semantic Union of the relations CAR(..., paint,type,...) and VEHICLE (... , color,type,...) is denoted: SemUnion ((paint,color),(type,type)).

Notation facility : When the semantic flexibility concerns only one pair of attributes, and if these attributes have the same names in the two relations, we will note shortly: **color-SemUnion** of the relations car (... ,color,...) and the relation vehicle (... , color,...). In the same way, we will note other operators: "**color-SimJoin**", "**color-SemRestriction**", "**color-Intersection**" and "**color-SemDifference**".

4.2 The Semantic Join Operator

The SemJoin operator is used for queries such as "Do people buy cover-seats and car having the same color?". The denominations of the colors for the cars and the cover-seats may be very different, since they have different manufacturers, each one having its own denominations. For this query, the user is not interested in an exact match of the color names. He just expects these color to be the same with respect to the priority levels of the linguistic dictionary.

Use of the linguistic Assertions in the SemJoin Algorithm

Let us consider the A_i -SemJoin of $R1(\underline{A_1}, \dots, A_i, \dots, A_n)$ and $R2(\underline{B_1}, \dots, B_{i-1}, A_i, B_{i+1}, \dots, B_m)$ on a criteria $R1.A_i = R2.A_i$ and P_i , where P_i is a set of other join predicates (e.g. $R1.A_3 = R2.B_4$)

For each matching comparison between $R1.A_i$ and $R2.A_i$:

- If $R1.A_i$ is equivalent to $R2.A_i$ then the matching succeeds,
- If $R1.A_i$ includes $R2.A_i$ then the matching succeeds,
- If $R1.A_i$ is included in $R2.A_i$ then the matching succeeds,
- If $R1.A_i$ and $R2.A_i$ are co-class then the matching succeeds,
- If $R1.A_i$ and $R2.A_i$ are dis-class then the matching fails.

We notice that the result of the SemJoin operator is strongly dependant on the priority level (the case "co-class" leads to a success while the case "dis-class" leads to a failure). This is coherent with the idea of introducing natural language in the queries, because the priority level exactly answer to the problem of what do we mean in the courant natural language by sentences like "do they have the same color?".

4.3 Semantic Restriction

The SemRestriction operator aims at answering queries such as "Have red cars more accidents than others ?", even if the stored colors for the car are not "red" but "Vermilion" or "Sevilla".

Use of the linguistic assertions in the SemRestriction Algorithm

Let us consider the A_i -SemRestriction on the $R(\underline{A_1}, \dots, A_i, \dots, A_n)$ relation with the criteria $R1.A_i = \text{val}A$,

For each matching comparison between $R1.A_i$ and $\text{val}A$:

- If $R1.A_i$ is equivalent to $\text{val}A$ then the matching succeeds,
- If $R1.A_i$ includes $\text{val}A$ then the matching fails,
- If $R1.A_i$ is included in $\text{val}A$ then the matching succeeds,
- If $R1.A_i$ and $\text{val}A$ are co-class then the matching fails,
- If $R1.A_i$ and $\text{val}A$ are dis-class then the matching fails.

We notice that the SemRestriction operator uses the hierarchy of the linguistic dictionary without considering the priority level. For example when a user wants all the vermilion cars, he obviously wants all the cars which colors are subsumed by vermilion, whatever the priority level is. Giving in the result the cars for which we

only know that they are red should imply to dissociate in the result the cars which "are" vermilion and the car which "may be" vermilion but also may be an other kind of red. In this work, we did not address this possibility.

4.4 The Semantic Union Operator

In a data warehouse system, the role of a union operator is to state whether two tuples correspond to the same object and to merge them in the resulting relation. The SemUnion operator can technically help in both. But in practice, it is particularly adapted to improve the second point, that is a correct merge of two tuples corresponding to the same object. For this purpose, the operator has to be applied on non key attributes. Then, an exact matching (by equality or through conversion procedure) is applied on the keys, assuming that the tuples correspond to the same object, while a linguistic matching is applied on other attributes for merging included or co-class values.

Use of the linguistic assertions in the SemUnion Algorithm

Let us consider the Ai-SemUnion of $R1(\underline{A1}, \dots, Ai, \dots, An)$ and $R2(\underline{A1}, \dots, Ai, \dots, An)$, for each pair of tuples from $R1 \times R2$ such that all A_j values are matching for $j \neq i$,

If $R1.Ai$ is equivalent to $R2.Ai$ then the two tuples are merged

If $R1.Ai$ includes $R2.Ai$ then the two tuple are merged, and the chosen value for Ai is $R2.Ai$

If $R1.Ai$ is included in $R2.Ai$ then the two tuples are merged, and the chosen value for Ai is $R1.Ai$

If ($R1.Ai$ and $R2.Ai$ are co-class) or ($R1.Ai$ and $R2.Ai$ are dis-class) then

if cardinality ($R1.Ai$) and the cardinality ($R2.Ai$) are monovalued

then the two tuples are not merged and are considered as inconsistent

else the two tuples are inserted in the result.

We notice that this algorithm uses not only the concept hierarchy but also additional links of the linguistic dictionary, namely the cardinality links.

An interesting use of this operator is to check out for the consistency of some inter-schema assertions stored in the meta database, namely the equivalence assertions between the attributes of the schemas. If two attributes are related by an assertion stating their semantic equivalence, and if none of their values are found linguistically equivalent, that probably means that the attributes have different meanings, and that the semantic equivalence assertion between them must be reconsidered.

4.5 Semantic Intersection and Semantic Difference

The SemIntersection and SemDifference operators are close to the SemUnion operator, and they use the linguistic knowledge in the same way. We just give in this section an example for each operator in Fig.5 and Fig. 6.

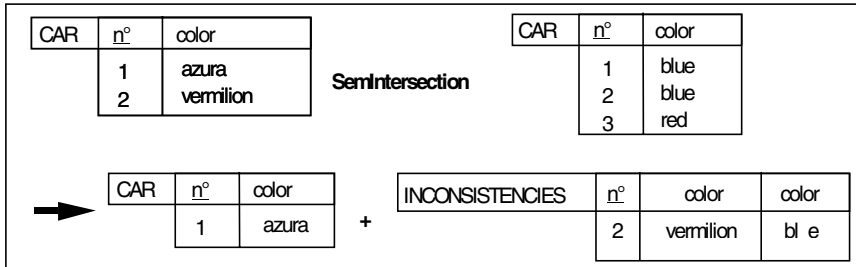


Fig. 5.: An example of SemIntersection

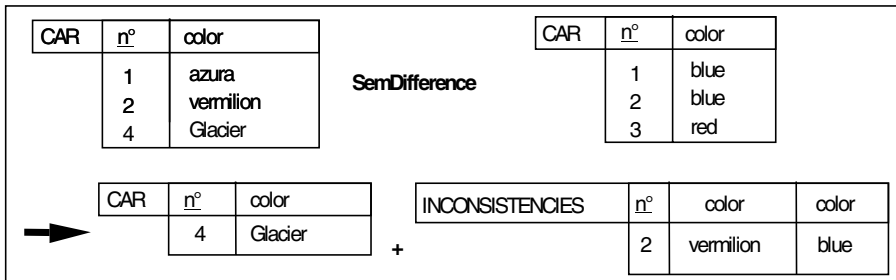


Fig. 6.: An example of SemDifference

5. Implementation of the Algorithms by the Way of Classical Algebra Operators

The semantic operators are computed by calling classical operators, after an automatic modification of the relations, by the mean of an embedded SQL. The implementation is performed using PL/SQL (for the ORACLE 8 DBMS). In this prototype, the linguistic dictionary is also stored using ORACLE 8.

Implementation of the Semantic Join

In order to compute the A-semantic join of two relations R1 (..., A,...) and R2 (...,A,...) on the criteria $R1.X = R2.Y$ where X is a set of attributes of R1 including A, and Y is a set of attributes of R2 including A, three steps have to be performed :

Step 1: A new attribute SemA is added to the schema of the R1 relation and a new attribute SemA is added to the schema of the R2 relation. Let us call these two new relations R1' and R2'.

Step 2 : The instances of R1' and R2' are computed by the following algorithm: each tuple r1 of R1 is translated into a tuple r1' of R'1 by filling the value for SemA with the priority node subsuming A in the linguistic dictionary. If no priority node is subsuming A, then the value is A itself. If several priority nodes are subsuming A, the r1 tuple gives several tuples in R1', each of them having a value for SemA corresponding to each priority node. The same process is applied o fill R2'.

Step 3: The semantic join between R1 and R2 on the criteria X=Y is transformed into a classical join between R1' and R2' with the criteria X' = Y', where X' = ((X U SemA) - A) and ((Y' = Y U SemA)-A).

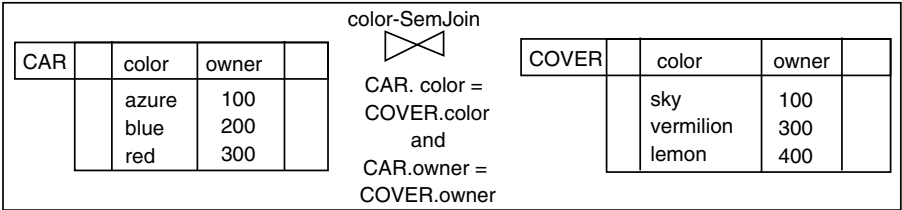


Fig. 7.: An example of semantic join

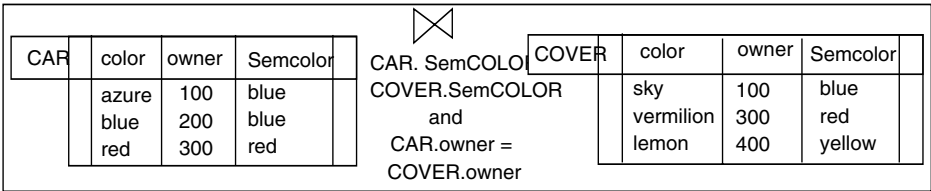


Fig. 8.: Transformation of the semantic join of figure 7 into a classical join

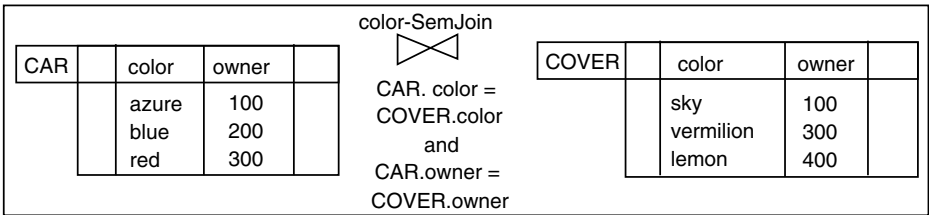


Fig. 9.: the rough result of the join, before projections

An example of query involving such join may be "how many people buy cars and seat-covers having the same color?". This is a typical marketing problem and the results may have an impact on further offers. Fig. 7 shows the semantic join, Fig. 8 shows the transformation of the semantic join into a classical join according to the three steps algorithm exposed above in this paragraph, and Fig. 9 shows the result of this classical join. Then a projection gives the set {100, 300} of persons having the same color for their cover-seats and their cars.

The optimization of the semantic operators execution is not address in this paper, but may be a weakness of the method. Obviously, if the semantic operators where implemented inside a DBMS, they could take profit of a panel of optimization

techniques and they would be more efficient. However, we do think that a main advantage of our method is to be compatible with the use of existing DBMS.

6. Conclusion

In this paper, we have proposed a method for dealing with the linguistic heterogeneity introducing some linguistic flexibility in the queries. We have defined the semantic operators and their implementation which uses classical operators applied on transformed relations. The linguistic metadata supporting these transformations is provided by a linguistic dictionary. We discussed the particularity of this dictionary with respect to a general linguistic dictionary as those used for schema integration.

Values in the dictionary are organized in a subsumption hierarchy in which some nodes are considered as "priority nodes". This means that each sub-tree having a priority node is the top represents a class of "linguistically close" values. The priority nodes are expected to be the most frequently referred denominations in the common language. They are used as parameters for allowing the user to define the scope of the flexibility in a very natural manner, without specifying any distance.

This approach introduces some linguistic flexibility in the queries, with a setting its scope in a well suitable for decision making tools over SIMS. They allow to deal with the high level concept values used in selling or statistic analysis such as "water", "coffee", "blue" while sources have recorded intermediate or basic values such as "plane water", "Lazarro", "azure" or "Glacier". Moreover, this technique contributing to a better understanding may be used for other contexts such as extraction on the WEB.

Acknowledgments

We wish to thank Mokrane Bouzeghoub for his valuable suggestions during this work.

References

1. Agarwal, S., Keller, A.M., Wiederhold, G. Krichna, S.: Flexible relation: an approach for integrating data from multiple, possibly inconsistent databases. Eleventh International Conference on Data Engineering, IEEE (1995)
2. Arens, Y., Knoblock, C.A., Shen, W.M.: "Query Reformulation for dynamic information integration. International Journal on Intelligent and Cooperative Information Systems (6) 2/3, (1996)
3. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Source Integration : conceptual modeling and reasoning support. Proceedings of the 6th Internat. Conf. In Cooperative Information Systems, (CoopIS'98) (1998)
4. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman J., Widom J.: The TSIMMIS project: Integration of Heterogeneous Information Sources", IPSJ'94 (1994)

5. Cluet, S., Delobel, C., Simeon, J., Katarzyna, S.: Your Mediators Need Data Conversion! . ACM SIGMOD, Seattle (1998)
6. DeMichiel, L.G.: Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. IEEE Transactions on Knowledge and Data Engineering, vol.1,n°4 (1989)
7. EDR Electronic Dictionary Technical Guide, Japan Electronic Dictionary Research Institute, Ltd. Mita-Kokusai-Bldg (1993)
8. Fankhauser, P., Kracker, M., Neuhold, E.J.: Semantic vs. Structural Resemblance of Classes. Sigmod Record, 20 (4) October (1991)
9. Fellbaum, C. et al., "WordNet : An electronic lexical database", MIT Press, ISBN 0-262-06197-X, (1998)
10. Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., Zhuge, Y.: The Stanford Data Warehousing Project. Bulletin of the Technical Committee on Data Engineering (1995)
11. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P., (edits): Fundamentals of Data Warehouses, Springer (1999)
12. Johannesson, P.: Logic-based approach to schema integration. Proc. of the 10th ER Conference (1991)
13. Johannesson, P.: Using conceptual graph theory to support schema integration. Proc. of the 12th ER Conf. (1993)
14. Kedad, Z., Bouzeghoub, M.: Conception de systèmes d'information multi-source. XVIIe Congrès Informatique des organisations et systèmes d'information et de décision (INFORSID) (1999)
15. Labio, W., Zhuge, Y., Wiener, J., Gupta, H., Garcia-Molina, H., Widom, J.: The WHIPS prototype for Data Warehouse creation and Maintenance, SIGMOD (1997)
16. Lenat, D.B. Guha, R.V., Pittman, K., Pratt, D. and Shepherd, M.: CYC: Toward Programs with Common Sens. Communications of the ACM, vol.33, n°8 (1990)
17. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Description. VLDB (1996)
18. Li C., Yerneni, R., Vassalos, V., Garcia-Molina, H., Papakonstantinou, Y., Valiveti, M.: Capability Based Mediation in TSIMMIS. IPSJ (1994)
19. Métails, E., Meunier, J.N., Levreau, G.: Database Schema Design, Validation and View Integration: a Perspective from Natural Language. Proc. of the 12th ER Conference (1993)
20. Métails, E., Kedad, Z., Comyn-Wattiau, I., Bouzeghoub, M.: Using Linguistic Knowledge in View Integration: toward a third generation of tools: in the review Data and Knowledge Engineering, vol. 23, n°1 Jun (1997)
21. Mirbel, I.: Semantic integration of conceptuel schemes" First International Workshop on Application of Natural Language to Data Bases (1995)
22. Resnik, P.: Using Information Content to Evaluate Semantic Similarity in a Taxonomy, IJCAT'95 (1995)
23. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous and autonomous databases. ACM Computing Surveys, Vol 22, n°3,September (1990)

24. Singh, M.P., Cannata, P.E., Huhns M.N., Jacobs, N., Ksiezyk T, Ong K, Sheth, A.P., Tomlinson, C., Woelk, D.: The Carnot Heterogeneous Database Project: Implemented Applications. In Distributed and Parallel Databases" Journal, vol. 5, n° 2, pages 207-225, April (1997)
25. Song, W.W., Johannesson, P., Bubenko, J.A.: Semantic similarity relations and computation in schema integration. in the review "Data and Knowledge Engineering", 19(1996)
26. Theodoratos D., Sellis T. "Data Warehouse Schema and Instance Design", ER'98, (1998)
27. Wiederhold, G., Genesereth, M.: The basis for mediation. proc. of CoopIS'95, (1995)
28. Zhou, G. Hull, R. King, R. Franchitti, J.C.: Data Integration and Warehousing using H2O. Bulletin of Technical Committee on Data Engineering (1995)

Detecting Redundancy in Data Warehouse Evolution

Dimitri Theodoratos*

Department of Electrical and Computer Engineering
Computer Science Division
National Technical University of Athens
Zographou 157 73, Athens, Greece
`dth@dblab.ece.ntua.gr`

Abstract. A Data Warehouse (DW) can be abstractly seen as a set of materialized views defined over a set of remote data sources. A DW is intended to satisfy a set of queries. The views materialized in a DW relate to each other in a complex manner, through common subexpressions, in order to guarantee high query performance and low view maintenance cost. DWs are time varying. As time passes new materialized views are added in order to satisfy new queries or for performance reasons while old queries are dropped. The evolution of a DW can result in a redundant set of materialized views.

In this paper we address the problem of detecting redundant views in a given DW view selection, that is, views that can be removed from the DW without negatively affecting the query evaluation or the view maintenance process. Using an AND/OR dag representation for multiple queries and views, we first provide a method for detecting materialized views that are not needed in the process of propagating source relation changes to the DW. Then, we use this method to detect materialized views that are redundant. As a side effect, our approach shows how source relation changes can be propagated to the DW materialized views by exploiting common subexpressions between views and by using other materialized views that are not affected by these changes.

1 Introduction

A data warehouse (DW) is a “subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making” [8]. DWs are designed for answering the queries of data-workers and analysts that are needed for decision support. A DW provides integrated access to multiple, distributed, possibly heterogeneous databases and other information sources: selected information from each source is extracted in advance, translated and filtered as appropriate, merged with relevant information from other

* Research supported by the European Commission under the ESPRIT Program LTR project “DWQ: Foundations of Data Warehouse Quality”

sources and stored in a repository. Current DWs contain very large volumes of data and support On-Line Analytical Processing (OLAP).

A DW can be seen as a set of *materialized views* defined over source relations. Some views may also be defined over other views. In order to ensure high query performance, the queries to the DW are answered by completely rewriting them [9] over the views stored in the DW. DW maintenance is performed by propagating source relation changes to the materialized views that are affected by these changes.

View maintenance. Usually the changes to the views are computed *incrementally* as opposed to their recomputation from scratch. In an incremental strategy, the changes to the views are computed using the changes to the source relations [4,2]. In order to compute the changes to the materialized views it may be necessary to issue queries against the data sources. We call these queries *maintenance queries*. When the source relation changes affect more than one materialized view, multiple maintenance queries are issued against the source relations for evaluation. The techniques of *multiquery optimization* [14] can be used to detect “common subexpressions” between these queries and to derive a global plan whose evaluation process is more efficient than a plan that evaluates each query separately. The evaluation cost of a maintenance query can be reduced if this query can be partially rewritten using a view already materialized. Often, materialized views are added to the DW exclusively for this purpose and are called *auxiliary views* [12,18]. By appropriately selecting auxiliary views to materialize in the DW, it is possible to maintain the initial materialized views and the auxiliary views altogether, for any source relation change, without issuing queries against the source relations. Such a view set is called *self-maintainable* [11].

DW design. The design of a DW involves selecting a set of views to materialize based on the set of queries that the DW need to satisfy. This choice is subject to a number of constraints and requirements. From a user viewpoint the query response time and the currency of the answer data for each query are of interest. These time periods should not exceed a certain limit specified by the user, or the data returned may be of no use. From a system viewpoint, the overall query evaluation cost and the overall view maintenance cost (or a combination of them), and the space needed for materializing the views are of interest.

DW evolution. DWs are time-varying. They are dynamic entities that evolve in time both in terms of schema and content.

As time passes new materialized views need to be added to a DW. There are three reasons for adding new materialized views:

- (a) New queries need to be satisfied by the DW. Then, new views are added that together with the existing views are able to answer the new queries [19].
- (b) Higher query response time is required for some queries. By adding new (more complex) views and by rewriting these queries using the new views the response time of the queries can be improved [1].
- (c) The view maintenance cost is high and the view maintenance process delays the evaluation of the queries; the query response time is unsatisfactory when

an “at query time” deferred maintenance policy is adopted (that is, a view is updated when a query involving this view is issued against the DW [13]); the currency of the answer data [15] is unsatisfactory or makes this data useless due to the excessive time needed to maintain the views over which the query is evaluated. By adding appropriately selected materialized views to the DW, the view maintenance cost can be reduced and/or the currency of answer data can be improved.

Also, as time passes, queries that the DW used to satisfy may become useless, either because the analysts do not need them anymore or because they are replaced by other more useful (and possibly more complex) queries.

1.1 The Problem

The augmentation of the DW by new views and the suppression of some queries can result in a materialized view set that contains redundant views. This can also happen with the initial design of the DW. Given a set of queries that are operational with the current design of a DW, a view V is redundant if:

- (a) V is not used in the optimal evaluation of a query, and
- (b) V is not used as an auxiliary view in the optimal propagation of source relation changes to the materialized views that are used for optimally evaluating the queries.

In this paper we address the problem of detecting redundant materialized views in a DW. This problem is complex because queries and views relate to each other through common subexpressions. For instance, the suppression of a query Q may suggest that a view V that was used for optimally answering Q is redundant. However, this is not true if V appears in the optimal evaluation plan for an operational query or if it is used as an auxiliary view to support the optimal maintenance of a non-redundant view.

Removing redundant views from the DW: (a) improves the availability of the system: no changes need to be applied to these views and the computation of changes for these views is possibly avoided, and (b) frees valuable space which can be used for storing other views and access structures that improve the evaluation of the queries and the maintenance of the views.

It is worth noting that “redundancy” does not refer to the intended redundancy in a DW for performance reasons.

1.2 Contribution and Outline

The main contributions are the following:

- We formalize the problem of detecting redundant materialized views in a DW based on a marked AND/OR dag representation for multiple queries and views. This formalization considers a large class of queries and views including grouping/aggregation queries, and applies to a generic DW architecture.
- Using multiquery AND/OR dags, we show how optimal query evaluation plans over the materialized views can be determined. We also show how the propagation of source relation changes to the materialized views can be

performed by taking into account common subexpressions between the views and by using materialized views that are not affected by these changes.

- We provide a procedure for determining materialized views that are useless in the propagation of source relation changes to the materialized views.
- Finally, we present a method for detecting redundant views in a given materialized view selection intended to satisfy a set of queries.
- Our approach is independent of the cost model used for evaluating queries and computing changes to the materialized views.

The rest of the paper is organized as follows. The next section presents related work. In Section 3, we specify the class of queries and views considered, and we introduce multiquery AND/OR dags. We base our analysis on a DW system architecture and operation presented in Section 4. In Section 5, we show how useless materialized views in the propagation of source relation changes can be determined. A method for detecting redundant materialized views in a DW is presented in Section 6. Finally, Section 7 contains concluding remarks.

2 Related Work

We are not aware of any research work dealing with the issue of non-intended redundancy in the design of a DW.

Answering queries using views has been studied in many papers, e.g. [9]. Materialized view maintenance has been addressed in recent years by a plethora of researchers. A number of papers dealing with different aspects of materialized view maintenance are cited in the introduction and in next sections. A nice overview of incremental view maintenance issues is provided in [5].

View selection problems for Data Warehousing usually follow the following pattern: select a set of views to materialize in order to optimize the query evaluation cost or the view maintenance cost, or a combination of both, possibly in the presence of some constraints. Given a materialized SQL view, [12] presents an exhaustive approach as well as heuristics for selecting auxiliary views that minimize the total view maintenance cost. In [6] greedy algorithms are provided for selecting views to materialize that minimize the query evaluation cost under a space constraint. A solution for selecting views that minimize the combined cost is given in [20]. A variation of the DW design problem endeavoring to select a set of views that minimizes the query evaluation cost under a total maintenance cost constraint is adopted in [7].

None of the previous approaches requires the queries to be answerable exclusively from the materialized views in a non-trivial manner. This requirement is taken into account in [17] where the problem of configuring a DW without space restrictions is addressed for a class of select-join queries. This work is extended in [18] in order to take into account space restrictions, multiquery optimization over the maintenance queries, and the use of auxiliary views when maintaining other views. Another extension of [17] deals with the same problem for a class of PSJ queries under space restrictions [16], while [19] addresses an incremental version of the DW design problem (dynamic DW design).

3 Multiexpression AND/OR Dags

In this section we define multiexpression AND/OR dags and their derivatives: multiquery AND/OR dags, query evaluation dags and change propagation dags. We start by specifying the class of queries and views considered here.

3.1 Class of Queries and Views

We adopt a natural extension of the relational algebra operations to bags (multisets). This algebra allows defining queries and views that have the SQL bag semantics. It contains the following operators: σ_C , where C is a selection condition (selection), Π_X where X is a set of attributes (projection), $\uplus$ (additive union), $\dot{-}$ (monus), $\min$ (minimal intersection), $\max$ (maximal union), ϵ (duplicate elimination), $\times$ (Cartesian product), $\bowtie_C$ where C is a join condition (conditional join), $\bowtie$ (natural join). The algebra includes also a grouping/aggregation operator: $\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}$ where X is a set of grouping attributes, agg_i is an aggregation function on attribute A_i , and B_i is a new name for the corresponding aggregated attribute. This operator is the generalized projection operator [3] extended to account for the naming of the aggregate attributes. As in SQL, the aggregation function agg can be `sum`, `avg`, `count`, `max`, `min`. Including a grouping/aggregation operator in the algebra is necessary in order to handle queries used in OLAP and Decision Support System (DSS) applications. The semantics of the previous operators are well known [2,3]. Queries and views considered here are expressions formed with these operators and relation or view names.

3.2 Multiquery AND/OR Dags

Alternative ways for evaluating an expression of those considered here can be compactly represented by an *AND/OR dag* [6]. A particular representation of AND/OR dags distinguishes between AND nodes and OR nodes [12]. We use here this representation for multiple queries, extended with marked nodes to account for views materialized at the DW [19,15].

Definition 1. An *expression AND/OR dag* for an expression e defined over a set of views (and/or relations) $\mathbf{V}$ is a rooted bipartite dag $\mathcal{G}_e$. The nodes of $\mathcal{G}_e$ are partitioned in AND nodes and OR nodes. An AND node is called an *operation node* and is labeled by an operator, while an OR node is called a *view node* and is labeled by a view. In the following we may identify nodes with their labels. An operation node has one or two outgoing edges to view nodes and one incoming edge from a view node. A view node has one or more outgoing edges (if any) to operation nodes and one or more incoming edges (if any) from an operation node. The root node and sink nodes of $\mathcal{G}_e$ are view nodes. The root node is labeled by e , while the sink nodes are labeled by views in $\mathbf{V}$.

Given a set of expressions $\mathbf{E}$ defined over a set of views $\mathbf{V}$, a *multiexpression AND/OR dag* $\mathcal{G}$ for $\mathbf{E}$ is an AND/OR dag resulting by merging the expression AND/OR dags for the expressions in $\mathbf{E}$. $\mathcal{G}$ is not necessarily a rooted dag (that is

it does not necessarily have a single root). All the root nodes of $\mathcal{G}$ are view nodes labeled by expressions in $\mathbf{E}$ (but not all the expressions in $\mathbf{E}$ label necessarily root nodes). In addition, view nodes in a (multi)expression AND/OR dag *can be marked*. Marked nodes represent views materialized at the DW. $\square$

We can now define query and multiquery AND/OR dags.

Definition 2. A *query AND/OR dag* for a query Q defined over a set of views and/or relations $\mathbf{V}$ is an expression AND/OR dag for Q .

A *multiquery AND/OR dag* for a set of queries $\mathbf{Q}$ defined over $\mathbf{V}$ is an expression AND/OR dag for $\mathbf{V}$. The view nodes representing (and labeled by) the queries in $\mathbf{Q}$ are called *query nodes*. $\square$

Example 1. Consider the set of views $\mathbf{V} = \{V_1, V_2, V_3\}$. Suppose that the schemes of their materializations are as follows: $V_1(A, B)$, $V_2(A, C)$ and $V_3(\underline{A}, B)$. An underlined attribute denotes the key of the corresponding view materialization. Consider also the set of queries $\mathbf{Q} = \{Q_1, Q_2, Q_3\}$ over $\mathbf{V}$ where:

$$\begin{aligned} Q_1 &= \pi_{A, \max(B) \text{ as } G}(\sigma_{B > 10}(V_1) \uplus \Pi_{AB}(V_2 \bowtie V_3)), \\ Q_2 &= \sigma_{A < 'c' \wedge E \leq B}(\pi_{A, \text{count}(B) \text{ as } D, \text{sum}(B) \text{ as } E}(V_2 \bowtie V_3)), \text{ and} \\ Q_3 &= \sigma_{E \geq B + 10}(\pi_{A, \text{count}(B) \text{ as } D, \text{sum}(B) \text{ as } E}(\sigma_{A < 'c'}(V_2) \bowtie V_3)). \end{aligned}$$

In Figure 1 a multiquery AND/OR graph for $\mathbf{Q}$ over $\mathbf{V}$ is shown. Operation nodes are depicted by small circles while view nodes are depicted by larger ones. $\pi_{A, \text{count}(B) \text{ as } D, \text{sum}(B) \text{ as } E}$ is abbreviated as π . Marked nodes (materialized views) are depicted by filled black circles. $\square$

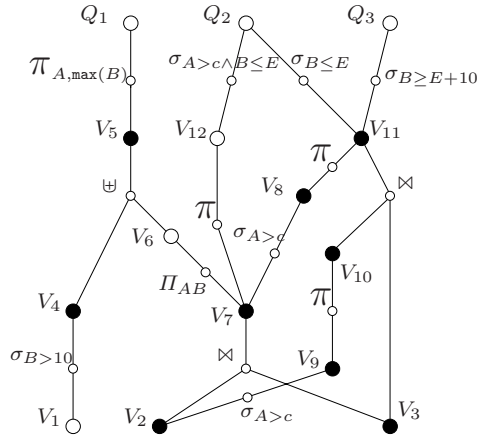


Fig. 1. A multiquery AND/OR dag for $\mathbf{Q} = \{Q_1, Q_2, Q_3\}$

Additional auxiliary definitions are provided below.

Definition 3. A *(multi)expression dag* is a (multi)expression AND dag (that is a (multi)expression AND/OR dag such that no view node has more than one outgoing edge).

A *subdag* $\mathcal{G}'$ of a (multi)expression AND/OR dag $\mathcal{G}$ is a (multi)expression AND/OR subdag of $\mathcal{G}$ such that:

- (a) if an operation node of $\mathcal{G}$ is in $\mathcal{G}'$, all its child view nodes in $\mathcal{G}$ are in $\mathcal{G}'$,
- (b) edges in $\mathcal{G}$ between nodes that are in $\mathcal{G}'$ are present in $\mathcal{G}'$, and
- (c) All and only the marked nodes in $\mathcal{G}$ that are present in $\mathcal{G}'$ are marked nodes in $\mathcal{G}'$. $\square$

3.3 Query Evaluation and Change Propagation Dags

Consider a multiquery AND/OR dag $\mathcal{G}$ for a set of queries $\mathbf{Q}$. A query evaluation plan over materialized views is represented by a query evaluation dag defined as follows.

Definition 4. A *query evaluation dag* for a query $Q \in \mathbf{Q}$ is an AND subdag $\mathcal{G}_Q$ of $\mathcal{G}$ such that:

- (a) $\mathcal{G}_Q$ is rooted at Q .
- (b) All and only the sink nodes of $\mathcal{G}_Q$ are marked nodes. $\square$

Example 2. Consider the multiquery AND/OR of example 1. Figure 5 shows query evaluation dags for the queries Q_1 , Q_2 and Q_3 . $\square$

A change propagation plan is represented by a change propagation dag defined below.

Definition 5. A *change propagation dag* for a sink node V in $\mathcal{G}$ is an AND subdag $\mathcal{G}_V$ of $\mathcal{G}$ such that:

- (a) All the marked view nodes that are ancestor nodes of V in $\mathcal{G}$ (that is the marked view nodes that occur in a path from a root node to V in $\mathcal{G}$) are present in $\mathcal{G}_V$, and the root nodes of $\mathcal{G}_V$ are among them.
- (b) A sink node in $\mathcal{G}_V$ is a marked node of $\mathcal{G}$, or node V . Node V is a sink node.
- (c) The non-sink marked nodes in $\mathcal{G}_V$ are ancestor nodes of V . $\square$

Example 3. Consider the multiquery AND/OR dag of example 1. Figure 2 shows two different change propagation dags for the sink node V_2 . Figure 6 shows change propagation dags for the sink nodes V_1 , V_2 and V_3 respectively. $\square$

4 DW System Framework

Our approach is based on a DW system whose basic architecture is depicted in Figure 3. The central component is the DW that contains a set of materialized views. Materialized views in the DW are depicted by small rectangles. The DW is built to satisfy a set of queries. These queries are issued by knowledge workers and analysts depicted at the top of the diagram. They are evaluated locally at the DW without accessing the remote data sources. Therefore, the DW contains

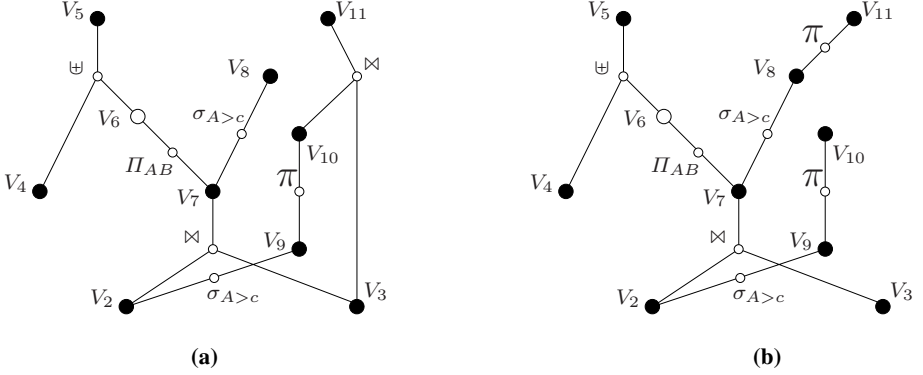


Fig. 2. Change propagation dags for the sink node V_2

materialized views that allow a complete rewriting of the queries over them. The bottom of the diagram shows the remote data sources that contain the source relations.

Source views. There is a set $\mathbf{V}$ of views each defined over the source relations of the same data source such that all the queries and the rest of the materialized views can be completely rewritten over $\mathbf{V}$. These views are called *source views*. Source views can be, for instance, source relations, select-project views over a source relation or even views defined over more than one relation from the same data source. In Figure 3 source views are depicted at the bottom of the DW component by small gray rectangles. Note though that source views may or may not be materialized views.

View maintenance. The DW views are maintained incrementally. We assume that the sources are aware of the corresponding source view definitions (if the latter are different than source relations) and are able to compute and send the changes to be applied to the source views, upon request from the DW or triggered by events. When the computation of the changes to a source view is performed at the source, transmission of useless change data is avoided and the availability of the DW is increased.

Changes for different source views are transmitted to the DW asynchronously. Upon arrival at the DW these changes are propagated to the affected materialized views. The views affected by the changes to a source view V are those that are defined using V . The affected views can be maintained separately. However, this process can be performed more efficiently if we propagate the source view changes to all the affected views together by exploiting (a) common subexpressions between the affected views and (b) other views materialized in the DW. Change propagation dags are used for this purpose. Their use in propagating changes to all the affected views together is described in the next section.

Type of changes. Concerning the type of changes these can be insertions and deletions (modifications are modeled by deletions followed by insertions). In

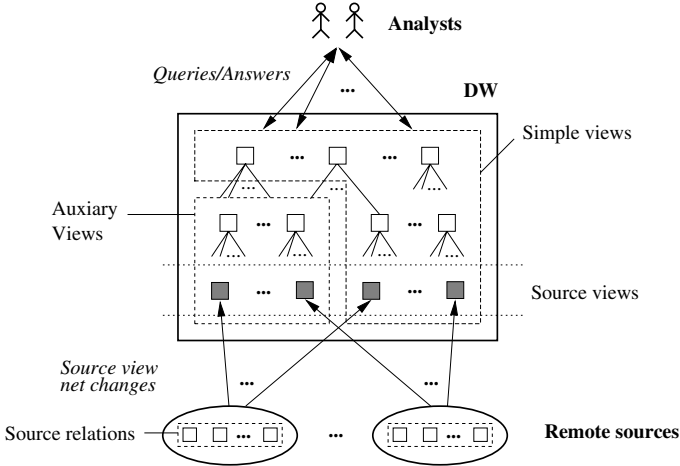


Fig. 3. A DW system architecture

order to avoid wasteful insertions and deletions (and data transmissions), when incrementally maintaining a materialized view, we consider only the changes actually inserted to or deleted from each view: the bag of tuples to be deleted from and that to be inserted to a view do not have any tuple in common, and a tuple to be deleted from a view occurs in the view materialization at least as many times as in the bag of tuples to be deleted from the view. These changes are called *net changes*. In the following by ‘changes’ we mean ‘net changes’.

Simple and auxiliary materialized views. The materialized views that are used for optimally answering the queries are called *simple views*. The rest of the materialized views are used for reducing the view maintenance cost of other materialized views, and are called *auxiliary views*. Simple views too can be used in the same manner, yet they have to appear in the optimal query evaluation plan of a query. A source view that is materialized can be either simple or auxiliary.

Self-maintainability. We assume that the set of materialized views in the DW is self-maintainable. Therefore, no maintenance queries against the source relations are needed for maintaining the materialized views.

5 Detecting Useless Views in a Change Propagation Dag

In this section we first show how changes to a source view can be propagated to the affected materialized views using change propagation dags. Then, we provide a procedure for detecting useless views in a change propagation dag.

5.1 Propagating Changes Using Change Propagation Dags

The changes to be applied to the materialized views that are affected by the changes to a source view V can be computed separately for each affected view.

The maintenance expressions provided in [2,10] allow for this computation. Consider a DW that satisfies a set of queries $\mathbf{Q}$. Let $\mathbf{V}$ be the set of source views, and $\mathcal{G}$ be a multiquery AND/OR dag for $\mathbf{Q}$ over $\mathbf{V}$ such that all the materialized views in the DW are marked view nodes in $\mathcal{G}$. $\mathcal{G}_V$ denotes a change propagation dag for a source view $V \in \mathbf{V}$. Recall that $\mathcal{G}_V$ includes all the materialized views that are affected by the changes to V . These views are ancestor view nodes of V in $\mathcal{G}$.

Changes are propagated in $\mathcal{G}_V$ bottom-up. This process starts with the source view node view V and then considers all its ancestor view nodes. Using the maintenance expressions in [2,10], changes to a view node are computed using the changes to its child view node(s). These expressions involve in general the pre-update state of the child view nodes (that is their state prior to the application of the changes), the pre-update state of the view node, and the changes to the child view nodes.

When the changes to a view node V' in $\mathcal{G}_V$ are computed, they are applied to it if V' is a marked node (materialized view). However, if the pre-update state of V' is needed in $\mathcal{G}_V$, this application is postponed until the changes and the pre-update state (if needed) of the parent view nodes of V' are computed. This way, the pre-update state of V' remains available when needed.

5.2 A Procedure for Detecting Useless Views

When computing the changes to a view node using the changes to its child view node(s) in a change propagation dag, the pre-update state of this view node and/or the pre-update state of its child view node(s) may not be needed.

Table 1 shows whether the pre-update state of a view V and the pre-update state of its child view node(s) V_1 (and V_2) are needed in the computation of the net changes to V , due to changes to V_1 , for different instances of the child operation node of V . These results can be derived from the maintenance expressions provided in [2,10]. Two cases for the monus operation are depicted ($V_1 \dot{-} V_2$ and $V_2 \dot{-} V_1$) since $\dot{-}$ is not a commutative operation. If V is a marked node, its pre-update state is available. Note that the table shows the pre-update states needed for the computation of both insertions and deletions. For instance, in the case of the monus operation $V_1 \dot{-} V_2$, only the pre-update state of V is needed for the computation of the deletions, while only the pre-update state of V_1 and V_2 is needed for the computation of the insertions to V ; therefore both pre-update states are shown as needed in Table 1. Note also that in the case of the grouping/aggregation operation $\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}$ where not all the aggregate functions agg_i are **min** or **max**, we consider that a **count** aggregate function is also computed by $\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}$, if **count** does not already appear in it. Further, since **avg** is computed in terms of **sum** and **count**, if $\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}$ includes an **avg** function, we assume that it also includes the corresponding function for **sum**.

If a view node is a marked node, its pre-update state is available. The pre-update state of a non-marked view node V is computed from the pre-update state of its child view node(s). This computation is not necessary if the pre-update state of V is not needed neither for the computation of the changes to V

V	pre-update state of V	pre-update state of V_1	pre-update state of V_2
$\sigma_C(V_1), \Pi_X(V_1)$	no	no	-
$\epsilon(V_1)$	no	yes	-
$\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}(V_1)$ where $agg_i = \text{sum}, \text{avg}, \text{or count}, i = 1, \dots, k$	yes	no	-
$\pi_{X, agg_1(A_1) \text{ as } B_1, \dots, agg_k(A_k) \text{ as } B_k}(V_1)$ where $agg_i = \text{min or max}, i \in [1, k]$	yes	yes	-
$V_1 \uplus V_2$	no	no	no
$V_1 \times V_2, V_1 \bowtie_C V_2, V_1 \bowtie V_2$	no	no	yes
$V_1 \min V_2, V_1 \max V_2,$	no	yes	yes
$V_1 \dot{-} V_2, V_2 \dot{-} V_1$	yes	yes	yes

Table 1. The need of the pre-update states of a view node V and of its child view node(s) V_1 (and V_2) for the computation of the net changes to V due to changes to V_1 .

nor by any of its parent view nodes. A parent view node of V may need the pre-update state of V for the computation of its own changes or (in case it is a non-marked view) because its own pre-update state is needed by one of its parent view nodes. Given a change propagation dag $\mathcal{G}_V$, we call *useless views* the view nodes in $\mathcal{G}_V$ whose pre-update state need not be computed.

Procedure *useless_view_nodes*, outlined in Figure 4, computes the useless view nodes in a given change propagation dag $\mathcal{G}_V$. In this process it uses the information provided in Table 1. *useless_view_nodes* proceeds in a top-down manner, from the root nodes of $\mathcal{G}_V$ to the sink nodes. It uses two set variables C and U . Variable C holds the view nodes in $\mathcal{G}_V$ that have already been considered, and is initially set to be the empty set. It is useful for not reconsidering a view node in $\mathcal{G}_V$ (and the view nodes in a subdag of $\mathcal{G}_V$ rooted at this node) when this is not necessary. Variable U holds the view nodes in $\mathcal{G}_V$ that are useless, and is initially set to contain the set of all the view nodes in $\mathcal{G}_V$. Procedure *useless_view_nodes* uses a recursive procedure, *check_needed*. *check_needed* is called on a view node and recursively calls itself on the child view nodes of this node. Initially, it is called by *useless_view_nodes* on the root nodes of $\mathcal{G}_V$. *check_needed* takes two arguments, W and P . W denotes the node on which *check_needed* is called, and P indicates whether W is needed for the computation of the changes to the parent view node of W from which it is called. The role of *check_needed* is to check whether W is needed given P . Note that a sink node in $\mathcal{G}_V$ is not needed for the computation of its own changes. Indeed, a sink node either it is not an ancestor node of V in $\mathcal{G}_V$ (and thus it is not affected by the changes to V), or it is node V . In the last case, V is not needed because we have considered that the corresponding source transmits to the DW the net changes to be applied to V . We assume that information on whether a view node is an ancestor node of V is kept with every node in $\mathcal{G}_V$. Also, we assume that the outgoing edges of a monus operation node are ordered in $\mathcal{G}_V$, in order to deal with the non-commutativity of this operation.

Input: a change propagation dag $\mathcal{G}_V$.

Output: the set U of useless view nodes in $\mathcal{G}_V$.

```

begin
     $C := \emptyset$ ;  $U := \{V' | V' \text{ is a view node in } \mathcal{G}_V\}$ ;
    for every root view node  $V_r$  in  $\mathcal{G}_V$  do
         $check\_needed(V_r, \text{true})$ 
    endfor;
    return  $U$ 
end.

procedure  $check\_needed(W, P)$ ;
begin
    if  $P = \text{true}$  or  $W$  is needed for the computation of its own changes then
        if  $W \notin C$  then
             $C := C \cup \{W\}$ ;  $U := U - \{W\}$ ;
            if  $W$  is marked then
                for every child view node  $W'$  of  $W$  do
                    if  $W'$  is needed for the computation of the changes to  $W$  then
                         $check\_needed(W', \text{true})$  else  $check\_needed(W', \text{false})$ 
                    endif
                endfor
            else /*  $W$  is not marked */
                for every child view node  $W'$  of  $W$  do
                     $check\_needed(W', \text{true})$ 
                endfor
            endif
        else /*  $W \in C$  */
            if  $W \in U$  then
                 $U := U - \{W\}$ ;
                if  $W$  is not marked then
                    for every child view node  $W'$  of  $W$  do
                         $check\_needed(W', \text{true})$ 
                    endfor
                endif
            endif
        endif
    else /*  $P = \text{false}$  and  $W$  is not needed for the computation of its own changes */
        if  $W \notin C$  then
             $C := C \cup \{W\}$ ;
            for every child view node  $W'$  of  $W$  do
                if  $W'$  is needed for the computation of the changes to  $W$  then
                     $check\_needed(W', \text{true})$  else  $check\_needed(W', \text{false})$ 
                endif
            endfor
        endif
    endif
end;

```

Fig. 4. Procedure *useless_view_nodes*

6 Detecting Redundant Views

Using the results of the previous section we can now detect redundant views in a DW. Consider a DW that satisfies a set of queries $\mathbf{Q}$ over a set of source views $\mathbf{V}$. A multiquery AND/OR dag $\mathcal{G}$ for this DW is a multiquery AND/OR dag for $\mathbf{Q}$ over $\mathbf{V}$ such that all the materialized views in the DW are marked view nodes in $\mathcal{G}$. Procedure *redundant_views*, presented below, proceeds in three steps, and computes a set of redundant views in $\mathcal{G}$.

Procedure *redundant_views*.

Input: A multiquery AND/OR dag $\mathcal{G}$.

Output: A set $\mathbf{R}$ of redundant views in $\mathcal{G}$.

1. - Compute the optimal query evaluation dags for the query nodes in $\mathcal{G}$.
 - Determine the simple views in $\mathcal{G}$. (Recall that a view is simple if it is a sink node of an optimal query evaluation dag.)
2. - Compute the optimal change propagation dag $\mathcal{G}_{V_i}$ for every source view V_i in $\mathcal{G}$.
 - For every $\mathcal{G}_{V_i}$ compute a subdag $\mathcal{G}'_{V_i}$ of $\mathcal{G}_{V_i}$ by removing from $\mathcal{G}_{V_i}$ all the nodes and edges that are not on a path from a simple view in every $\mathcal{G}_{V_i}$.
3. - For every $\mathcal{G}'_{V_i}$ use procedure *useless_view_nodes* to compute the set U_i of useless view nodes in $\mathcal{G}'_{V_i}$.
 - Let M_i be the set of marked view nodes of $\mathcal{G}$ in U_i that are not simple views.
 - Add to every M_i all the marked view nodes in $\mathcal{G}$ that are not in $\mathcal{G}'_{V_i}$ and are not simple views.
 - Return $\mathbf{R} = \cap_i M_i$ □

Example 4. Consider the multiquery AND/OR dag $\mathcal{G}$ for a DW shown in Figure 1. We apply procedure *redundant_views* to $\mathcal{G}$.

Suppose that step 1 computes the optimal query evaluation dags depicted in Figure 5. Then, the simple view nodes in $\mathcal{G}$ are V_5 and V_{11} . Suppose that

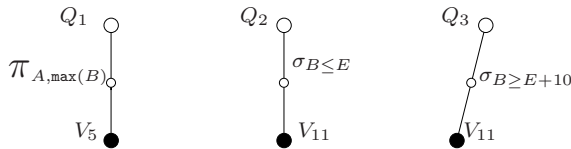


Fig. 5. Optimal query evaluation dags for the queries Q_1 , Q_2 and Q_3 .

step 2 computes the optimal change propagation dags $\mathcal{G}_{V_1}$, $\mathcal{G}_{V_2}$, and $\mathcal{G}_{V_3}$ for the source views V_1 , V_2 and V_3 respectively, depicted in Figure 6. Consider the optimal change propagation dags $\mathcal{G}_{V_2}$ and $\mathcal{G}_{V_3}$. The node V_8 along with the edges on the path from V_8 to V_7 do not appear in a path from a simple view

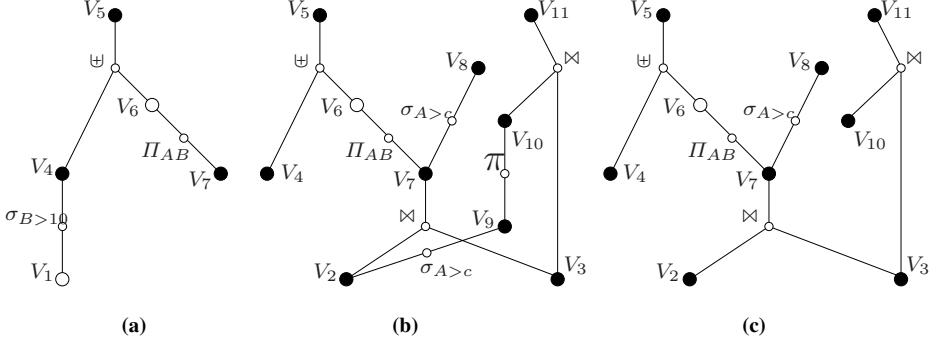


Fig. 6. Optimal change propagation dags for the source views (a) V_1 , (b) V_2 , and (c) V_3

in $\mathcal{G}_{V_2}$ and $\mathcal{G}_{V_3}$ and do not appear at all in $\mathcal{G}_{V_1}$. Therefore, this node and these edges are removed from $\mathcal{G}_{V_2}$ and $\mathcal{G}_{V_3}$ before proceeding to step 3. In step 3, procedure *useless_view_nodes* on the transformed change propagation dags first computes the sets $U_1 = \{V_1, V_4, V_5, V_6, V_7\}$, $U_2 = \{V_2, V_4, V_5, V_6, V_7, V_9, V_{11}\}$, and $U_3 = \{V_3, V_4, V_5, V_6, V_7, V_{11}\}$. Then, the sets $M_1 = \{V_2, V_3, V_4, V_7, V_8, V_9, V_{10}\}$, $M_2 = \{V_2, V_4, V_7, V_8, V_9\}$, and $M_3 = \{V_3, V_4, V_7, V_8, V_9\}$ are computed. Finally, the set $\mathbf{R} = \{V_4, V_7, V_8, V_9\}$ of redundant views is returned. $\square$

In step 2 of the procedure *redundant_views*, auxiliary views that are not on a path from a simple view in any optimal change propagation dag are not considered since they cannot be used in propagating changes to a simple view.

A view $V' \in \mathbf{R}$ is a materialized view such that V' is not a simple view and: (a) for every optimal change propagation dag $\mathcal{G}_V$, V' is useless in $\mathcal{G}_V$, or V' does not appear in $\mathcal{G}_V$, or (b) V' cannot be used in propagating changes to a simple view. Therefore, the following proposition can be shown.

Proposition 1. *Procedure *redundant_views*, applied to a multiquery AND/OR dag for a DW, correctly computes the set $\mathbf{R}$ of redundant views in this DW.* $\square$

The approach presented above for detecting redundant views is independent of the cost model used. The computed set of redundant views though depends on this model.

7 Conclusion

In this paper we have addressed the problem of detecting redundant views in a DW. This problem is evolved because queries and views relate to each other in a complex manner through common subexpressions. To this end, we have used a marked AND/OR dag representation for multiple queries and views. We have considered a large class of queries and views including grouping /aggregation queries that are extensively used in DW applications. We provided a procedure for determining materialized views that are useless in the propagation process of source relation changes to the materialized views. This procedure is used for

designing a method that detects redundant views in a given materialized view selection. Our approach applies to a generic DW system framework and does not assume a specific query evaluation and view maintenance cost model. We have also shown how the propagation of source relation changes to the materialized views can be performed by taking into account common subexpressions between the views and by using other views materialized in the DW.

An interesting extension of the present work would consider not only the optimal but also non-optimal change propagation plans in order to specify a set of views whose removal from the DW minimizes the overall view maintenance cost. This approach though may increase the cost of specific change propagation plans and it is not appropriate when data currency constraints are imposed.

References

1. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing Queries with Materialized Views. In *Proc. of the 11th ICDE*, pages 190–200, 1995. 341
2. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1995. 341, 344, 349, 349, 349
3. A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proc. of the 21st VLDB*, pages 358–369, 1995. 344, 344
4. A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1993. 341
5. A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *Data Engineering*, 18(2):3–18, 1995. 343
6. H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proc. of the 6th Intl. Conf. on Database Theory*, pages 98–112, 1997. 343, 344
7. H. Gupta and I. S. Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Proc. of the 7th Intl. Conf. on Database Theory*, 1999. 343
8. W. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 2nd edition, 1996. 340
9. A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM Symp. on Principles of Database Systems*, 1995. 341, 343
10. D. Quass. Maintenance Expressions for Views with Aggregation. In *Workshop on Materialized Views: Techniques and Applications*, pages 110–118, 1996. 349, 349, 349
11. D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self Maintainable for Data Warehousing. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, pages 158–169, 1996. 341
12. K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 447–458, 1996. 341, 343, 344
13. N. Roussopoulos and Y. Kang. Principles and techniques in the design of ADMS $\pm$. *Computer*, Dec 1986. 342
14. T. K. Sellis. Multiple Query Optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988. 341

15. D. Theodoratos and M. Bouzeghoub. Data Currency Quality Factors in Data Warehouse Design. In *Proc. of the Intl. Workshop on Design and Management of Data Warehouses*, Heidelberg, Germany, June 1999. 342, 344
16. D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the Global Data Warehouse with SPJ Views. In *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering*, Springer-Verlag, LNCS No 1626, pages 180–194, 1999. 343
17. D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997. 343, 343
18. D. Theodoratos and T. Sellis. Data Warehouse Schema and Instance Design. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling*, Springer LNCS 1507, pages 363–376, 1998. 341, 343
19. D. Theodoratos and T. Sellis. Dynamic Data Warehouse Design. In *Proc. of the 1st Intl. Conf. on Data Warehousing and Knowledge Discovery*, Springer-Verlag, LNCS, 1999. 341, 343, 344
20. J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 136–145, 1997. 343

Modelling Data Warehouses and OLAP Applications by Means of Dialogue Objects

Jana Lewerenz^{1*}, Klaus-Dieter Schewe², and Bernhard Thalheim¹

¹ Department of Computer Science, Brandenburg Technical University of Cottbus

² Technical University of Clausthal Computer Science Institute

Abstract. The idea of data warehouses is to provide condensed information in order to support managers in the analysis of business facts such as sales, costs, profits, etc. along various dimensions such as geography, organisation, time, etc. The analysis should allow fast switches between different selected multiple dimensions at different granularity. The task itself is usually called on-line analytical processing (OLAP). We show in this paper how to model data warehouses with OLAP functionality by means of *dialogue objects*. These are extended and possibly materialised views on collections of operative databases and couple structural and behavioural aspects of application units.

1 Introduction

Most information systems in practice have been designed and developed for the purpose of supporting operative tasks. Users invoke actions on selected data which lead to transactions on underlying databases. Thus, from a database point of view a central issue of such systems is on-line transaction processing (OLTP) although this term does not capture quite correctly the user's perspective [3]. Technically, OLTP is reflected in database design by putting emphasis on desirable properties such as non-redundancy and anomaly avoidance as realised by various normal forms.

For the design of management support systems that often involve on-line analytical processing (OLAP), these guidelines do not apply at all since the dominant aspect of managerial work is to ask for a condensed overview on business progress. Most access is read-only, involves large amounts of data and must be repeatable. User needs play a decisive role for information system design [14]. Nevertheless, OLAP applications can be considered as information systems requiring specialised functionality. This functionality can be provided by the combination of existing technologies.

1.1 Stars and Snowflakes

More precisely, a manager may ask for a fast overview on mostly numerical business facts such as sales, costs, profits, etc. over some period of time, e.g., a

* Supported by the German Research Society (DFG grant no. GRK 316)

week, a month or a year. These facts should be organised along different multiple dimensions such as geography (country, state, city, etc.), organisation structure (head quarters, agency, department, etc.), product structure (individual, group, kind, etc.) and many others.

The idea of a data warehouse is to provide a separate database to store these data. Thus, its structure can be described by a very simple entity-relationship schema with entity types for each dimension and a single n -ary relationship type for the facts. Such a schema is usually called a *star schema* [4,7,10]. Besides various other arguments, the use of star schemata underlines the conceptual need for multi-arity relations [17]. Due to the dominance of read-only access there is no need to consider normalisation.

At a more detailed level a dimension can be fanned out into a hierarchy. For example, a product dimension may include category, brand, package size, etc. In terms of the higher-order entity-relationship model (HERM) [17] only the largest of these ‘new’ dimensions will lead to an entity type; all others will be turned into (possibly unary) relationship types. Accordingly, the fact relationship type will now relate relationship types, i.e., it is a higher-order type. The resulting schema is usually called a *snowflake schema* [4,10]. Besides other arguments, snowflakes underline the advantages of higher-order relationship types [17]. Note that some authors, e.g., Kimball [7], negate the necessity to model snowflakes.

The data in instances of a warehouse schema—be it a star or a snowflake—stem from different operative databases. We may regard the collection of involved databases as a non-integrated virtual multi-database containing homonyms, synonyms and redundant data possibly organised in several different ways. There is, however, a transformation which maps an instance of this multi-database to an instance of our warehouse schema so that the warehouse turns out to be a view (or superview) [9,11] in the most general sense.

Even if we assume a copy semantics, i.e., data in operative databases will be time-stamped and added to the existing data in the view, this will not cause a significant change. The basic assumption of data warehouses being organised as separate databases then turns into the call for view materialisation. Whether this demand is generally justified or not is a matter of debate. It definitely depends on the time complexity of building the view and the frequency of access. Furthermore, note that the intended OLAP applications will almost always access small portions of the warehouse data that are themselves aggregated. This leads to views over a view which by compositionality are again views.

1.2 OLAP Functionality

The warehouse only provides the data to be processed. The major intention is, however, the support of managerial work by the means of OLAP functionality [6,12,19]. At the heart, this means to provide functions

- to present aggregated numerical data in various ways,
- to quickly switch between different presentations or to keep several presentations at a time,

- to quickly switch between different sections of data by means of changes in granularity (drill down or roll up), selected dimensions, selected facts, etc.,
- to process hypothetical changes, or
- to provide analytical models for the evaluation of data.

Of course, this list is not complete. It depends on the requirements of the management support system.

It is often claimed that database technology does not provide the functionality required by OLAP applications [7]. We will demonstrate below that the entity-relationship model provides the complete functionality and support necessary for OLAP applications. We observe that OLAP schemata can be regarded as external ER views with/without materialisation. OLAP schemata can be defined on the basis of QBE or SQL extensions for ER models such as HERM [18]. The presentation of data can be varied according to presentation rules of dialogue objects.

In fact, the difference between managerial and high-skilled clerical work is not as large as it appears in the literature [7,10]. Both applications share the use of underlying external views as emphasised above as well as the user-driven invocation of actions on the view data. In both cases it is, therefore, desirable to couple views with operations on them. This is the core idea of dialogue objects, a notion coined in the context of cooperative information system development [13] that has been thoroughly formalised [15].

Dialogue objects provide flexibly handable units for on-line processing. In both clerical and managerial work fundamental decisions on selecting work steps are left to the user. The major difference is that in the first case actions usually lead to queries or transactions on some operative database, whereas in the latter case they almost always lead to queries on the warehouse view. A challenge in OLAP is that even parameterisation of these queries should be supported. This may also have an impact on operative dialogue oriented systems.

In the remainder of this paper we shall present some more details to support our argumentation. Methodologically, this implies a change of focus in the design of management support systems providing OLAP functionality. The central question concerns the dialogue units required in managerial work. Structurally, this involves the definition of external views. Behaviourally, it provides the functions sketched above.

2 Modelling the Warehouse

We start with a more detailed discussion of structural aspects but keep the presentation rather informal. Throughout this section we use a simplified version of the grocery store example from Kimball [7] to illustrate our ideas. We shall first see how to model the application star schema using the entity-relationship model. We demonstrate that ER is well-suited for warehouse modelling contradicting statements made in some warehouse publications [7]. Then we show how to obtain the star schema as a view in the case of a single operative database.

The more realistic case of several operative databases appears as a simple generalisation. Finally, we handle the case of a snowflake schema using dimension hierarchies.

Efficient algorithms for the translation of HERM views into bundles of relational views exist [18] and can then be used for a realisation of those views.

2.1 The Grocery Store Example

An OLAP application. Suppose we want to analyse the development of sales in a grocery chain. We might, e.g., want to know how many items of certain products have been sold in certain stores or regions over some period of time, what the corresponding sale was and how much profit we made. So the facts we are interested in are **Quantity**, **Money_sales** and **Profit**. The dimensions are **TIME**, **PRODUCT**, **CUSTOMER** and **SHOP**.

Then the facts **Quantity**, **Money_sales** and **Profit** appear as attributes of the relationship type **PURCHASE**. In addition, we may have attributes **PID**, **Description**, **Category** for the entity type **PRODUCT**, attributes **SID**, **Town**, **Region**, **State**, etc. for the entity type **SHOP**, attributes **TID**, **Weekday**, **Month**, **Quarter**, etc. for the entity type **TIME** and attributes **Name**, **Address**, **Category**, etc. for the entity type **CUSTOMER**.

Note that a relational transformation usually will turn key attributes of the dimension types ($\times$ ID) into foreign keys for a fact table. Figure 1 shows the corresponding (simplified) ER schema in HERM notation [18].

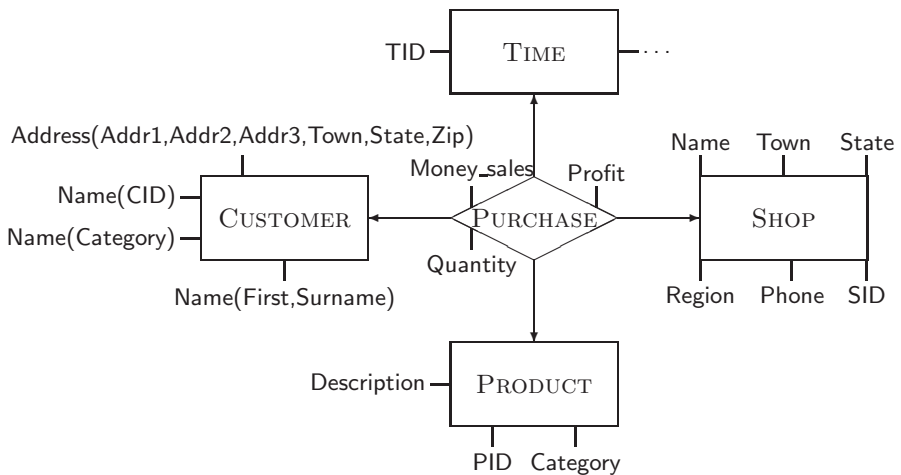


Fig. 1. Grocery Example: Star Schema on PURCHASES

The operational database. The data to be stored in instances of the warehouse schema mostly stems from operative databases. So assume that each purchase in a store will be registered together with the date and the number of items sold.

In addition we keep a price relation also depending on the date. For the moment we leave aside any further decomposition for products and stores. The date can be assumed to be composed of **day**, **month** and **year**. This is usually sufficient for operative purposes. In order to relate a date to the more complex data needed in the entity type **TIME** in the star schema, we may assume a general pre-determined time table containing all the time attributes needed there. Of course, (**day**, **month**, **year**) will appear as a composed key.

Figure 2 shows an ER schema for such a simple sales database (omitting some attributes). Note that this schema is not yet normalised, but for the sake of simplicity we abstain from further decomposition.

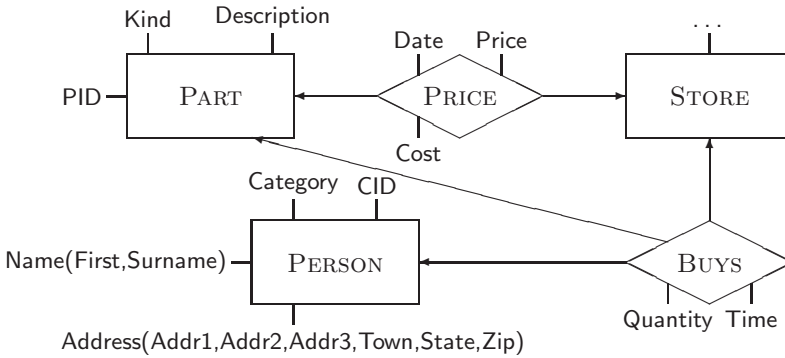


Fig. 2. Grocery Store: Operative Sales Schema

2.2 External Views

The central claim made in the introduction is that the star schema occurs as an external view on the operative sales schema (enriched by the assumed time table). In general, a view is nothing but a stored query, hence consisting of an input schema $\mathcal{S}_{in}$, an output schema $\mathcal{S}_{out}$ and a database transformation V mapping instances of $\mathcal{S}_{in}$ to instances of $\mathcal{S}_{out}$. Additionally, representation, formation and wrapping rules [2] can be added to the view depending on the dialogue object.

In our example $\mathcal{S}_{in}$ is the (enriched) sales schema (Figure 2) and $\mathcal{S}_{out}$ is the star schema (Figure 1). Since ER schemata—even higher-order schemata—are basically hierarchical in the sense that relationship types are based on entity types, it is sufficient to employ sequences of **SELECT-FROM-WHERE** statements to define the mapping V .

Different views can be defined for the application sketched above. The view displayed in Figure 1 is a star schema based on a relationship type. This star schema can be obtained via the following view definition:

```

create view StarSchema as
  Product : select PID : p.PID , Category : p.Kind ,
            Description : p.Description
    from Part p ,
  Time : select Time
    from Buys
  Customer : select Name : c.Name, CID : c.CID,
                  Address : c.Address, Category : c.Category
    from Person c
    where ... Category ...
  Shop : select ...
    from Store s
  Purchase : select Customer(CID) : ..., Product(PID) : ...,
                  Time(TID) : ..., ..., Shop(SID) : ...,
                  Money_sales : ..., Quantity : ..., Profit : ...
    from Buys(Part, Person, Store), Price pr
    where ... ;

```

In the same fashion the snowflake schema displayed in Figure 3 (partially without attributes) can be generated on the schema discussed in Section 2.1.

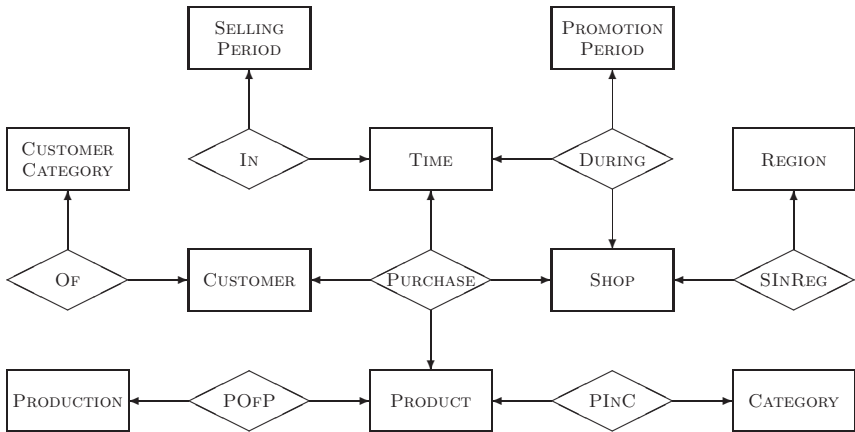


Fig. 3. Grocery Store: Snowflake Schema on Purchases

2.3 Dimension Hierarchies

In many cases it is desirable to consider more complex warehouse schemata than the flat star schema. This situation usually occurs if dimensions are organised hierarchically. E.g., products can be grouped into BRAND and shops into REGION. Using the higher-order ER model, BRAND and REGION will now occur as

entity types and the original types will be turned into unary relationship types. For the case of our grocery store example, this situation is illustrated by the snowflake schema in Figure 4, omitting all attributes.

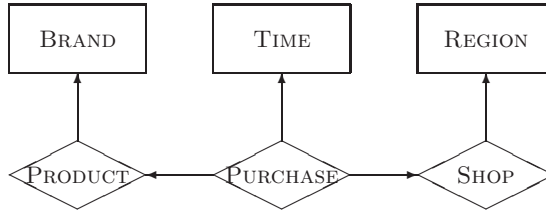


Fig. 4. Grocery Store: Snowflake Schema with Dimension Hierarchies

In a number of OLAP applications similar hierarchies for the presentation of time, products, people, organisations, addresses, etc. are used. We can define ER schemata for hierarchies. The corresponding databases may be instantiated automatically. The schemata can be used for the generation of hierarchic view sets.

OLAP schemata are then views defined on the basis of the application ER schema and on ER schemata for hierarchies. In this case, the snowflake schema displayed in Figure 3 is generated on the basis of the application discussed in Section 2.1 and on view types which are defined on hierarchy schemata for time, addresses, customer categories and product categories.

Time is modelled in OLAP applications on the basis of universal relation assumptions. The representation of time defined for the higher-order ER model uses several relationship types. The OLAP representation in Figure 5 is based on the universal relation approach. The OLAP type has 25 attributes. Additionally, an identifier attribute may be introduced. A large number of integrity constraints must be considered. The time dimension is necessary in OLAP applications because it facilitates slicing of data by workdays, holidays, fiscal periods, seasons and by major events. The time dimension is guaranteed to be present in every warehouse application, because virtually every data warehouse depends on a time series. Time is usually the first dimension for sorting.

The representation of time (where some information can be derived but is still explicitly contained) shows that OLAP schemata are redundant. Denormalisation is common in OLAP schemata. Understanding OLAP schemata as views with redundant representation and denormalisation does not create any problem.

Based on the roll-up and drill-down functions discussed below it is possible to display data in different granularity. This approach can be simulated by families of views. The family is generated in dependence of a given generalisation hierarchy. For instance, given the hierarchies

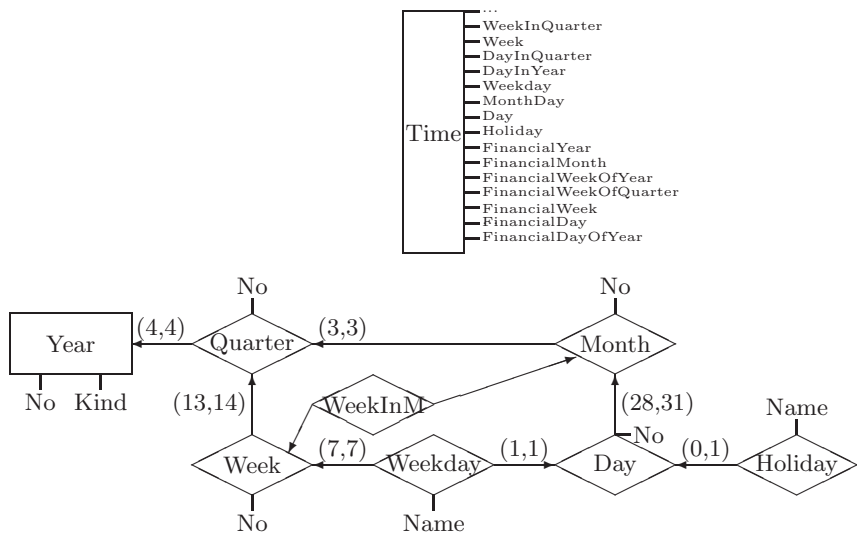


Fig. 5. Extended HERM Schema and OLAP Representation of Time

$Product \preceq Brand \preceq Manufacturer$ and $Shop \preceq Area \preceq Region$, we can create a query on shops and products. This query can be contracted according to the hierarchies to queries on brands, regions, etc.

With regard to being defined as views on operative database schemata, snowflake schemata do not add additional complexity since they simply correspond to some kind of normalisation along functional or multi-valued dependencies. The same normalisation can be expected in the underlying operative databases.

3 Modelling OLAP Applications

Now we turn to the problem of how to realise OLAP functionality in management support systems. Since OLAP applications constitute particular dialogue oriented information systems we should preserve the technology developed for such information systems [2,13,14,15] as much as possible.

We start looking at the presentation of facts in sections of the data warehouse followed by an investigation of typical actions involved in such presentations. This will lead us to a short discussion of dialogue objects and their suitability for OLAP. Finally, we discuss the tuning of presentation granularity which will constitute further operations.

3.1 Presentation of Facts

The typical OLAP scenario assumes a manager to query the warehouse. Since familiarity with sophisticated query languages cannot be presumed, such a query should be expressible in terms understandable to the user, i.e.:

- The manager selects the facts (s)he wants to consider. E.g., in our grocery store example (s)he may choose to look only at money_sales perhaps with a percentage-based comparison to an earlier period.
- The manager selects the dimensions and their granularity along which the selected facts should be presented. In our grocery store example this can be time, e.g., on the basis of last year's quarters, individual products and regions.
- In addition to the dimensions a specification of their relevant descriptive attributes is needed. E.g., it may be sufficient to get the product name and the grammalogues for the region and quarter.
- The manager states additional selection criteria to restrict the data to be presented. E.g., only stores that belong to a certain category (cheap or expensive equipment, etc.) or products offered in the most recent promotion campaign should be considered.
- Criteria for the grouping and ordering of the facts and dimension attributes should be given.
- Finally, the presentation of the result should be specified. E.g., tabular or graphical presentations such as scroll lists, beam or tart diagrams should be available.

The data to be presented constitute a view on the warehouse. Since we argued that the warehouse itself can be realised as a view on the operative multi-database, each data selection in an OLAP application constitutes also a view on the operative databases. Whether the intermediate warehouse level and maybe also other views are materialised or not depends on technical aspects. The manager need not be aware of any connection to the underlying operative databases.

(S)he may not even be aware of snowflake schemata if these have been designed, since the star schema is also a view on the snowflake schema. Finally, whether the star schema is presented to managers directly by the entity-relationship schema or simply by the list of available facts, dimensions, larger dimensions and dimension attributes, can be left to the manager's preferences.

Since the queries underlying the views are conjunctive a simple QBE like entry form will be sufficient. Thus, it is recommended to couple the view with actions to select facts, dimensions, dimension attributes, to add restricting conditions and to choose presentation preferences. We may therefore consider the presented data together with the actions on them as describing an object, namely a *dialogue object* (d-object).

More formally, a dialog object (d-object) consists of a unique abstract identifier, a set of values $v_1, \dots, v_n$ in associated fields $F_1, \dots, F_n$ which correspond to describing values of d-objects, a set of references to other d-objects in order to allow quick navigational access, a set of actions to change the data and to

control the dialogue, and a state with the possible values ‘active’ and ‘inactive’. This means that d-objects only exist as long as they are visible on the screen. If a window is closed the corresponding d-object is deleted.

The identifier serves the purpose of administrating d-objects. It is not known to and cannot be used by the user and is not visible. Only the active d-object allows manipulations of the represented data and only its actions can be invoked.

Users invoke actions to change selection criteria, to navigate to another (possibly new) dialogue object or to a modified presentation of the same dialogue object. This kind of system usage constitutes the basic object-action-principle in dialogue systems. Users enter or select values on the screen and invoke actions—usually grouped in menus—on them. The dialogue system reacts by offering other data or by activating and deactivating entries in selection lists or possible actions in the action bar. In graphical user interfaces data are normally presented in a window.

Once we know that the data in a d-object may be used in the next dialogue step, it may be helpful to provide also hidden data. Their presence may quicken the access to the database or the navigation to other d-objects. Depending on selections or entries made in a d-object only some of the possible actions may be allowed. The processing of an action may require further preconditions depending on the state of the dialogue system especially on other users’ d-objects.

Note that the selection of grouping, ordering and screen presentations does not affect the dialogue object itself. Thus, presentational aspects can be treated separately from the core of data processing.

3.2 Dialogue Types

Conceptually, we are not interested in individual d-objects but want to classify them into types. We shall, therefore, talk about *dialogue types* (d-type). Dialogue types unify structural, behavioural and presentational aspects of an application by combining view definitions, action specifications and rules for presenting an dialogue object’s contents to the user [16].

Structural aspects. At the heart of such a d-type we provide a view consisting of a (higher-order) ER schema and a defining query. The schema is the output schema $\mathcal{S}_{out}$ mentioned in the previous section; the defining query is the sequence V of query language statements which create instances of $\mathcal{S}_{out}$. The input schema $\mathcal{S}_{in}$ can be omitted. It is generally assumed that the warehouse schema—star or snowflake schema—is taken for this purpose. The view definition may be parameterised. Parameters are either specified as defaults in the d-type definition or can be modified by the user during interaction.

In addition, as indicated above, we may choose a subset of the query result as the data to be actually presented while keeping the rest for fast support of operation. This constitutes the *visual schema* $\mathcal{VS}_{out}$ as a subschema of $\mathcal{S}_{out}$. The view definition of the visual schema may again be parameterised. *Visibility functions* determine the respective parameters on the basis of default definitions, actions invoked by the user and/or explicit user specifications.

Building upon the grocery store warehouse from Figure 1 the d-type may, e.g., involve money sales per shop and region, but only the money sales per region will be presented to the user. Then the output schema $\mathcal{S}_{out}$ could be as presented in Figure 6 and the visual schema $\mathcal{VS}_{out}$ would be the subschema in which the relationship type SALES_PER_SHOP and the entity type SHOP have been omitted (cf. Figure 6). We abstain from a detailed description of the defining query which would use the star schema in Figure 1 as the input schema $\mathcal{S}_{in}$.

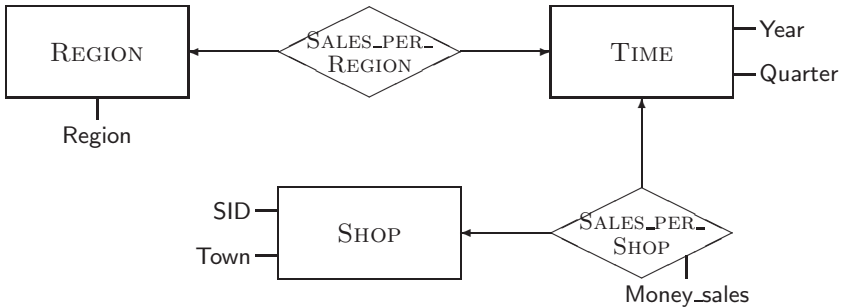


Fig. 6. Grocery Store: Schema underlying a D-Type

Behavioural aspects. Attached to the view we provide a collection of operations such as those discussed in Section 3.3. Such actions allow the user to navigate within the view (e.g., to formerly hidden parts), to navigate between views and to switch between different presentations of data. In addition to typical OLAP actions as described below, help functions can be made available, too.

Similar to the structural part that contains both visible and hidden data, dialogue actions are not necessarily accessible at all times. Availability of actions is controlled by action *access functions* which determine availability on the basis of the user's access rights, etc. For instance, based on the example in Figure 6, adding the fact *Money_sales* from the entity type SALES_PER_SHOP might only be possible for managers on a high enough level of the enterprise's hierarchy. Similar restrictions could exist for drill-down operations in certain dimensions.

To guarantee stable performance of the application *exceptions* are defined for the case that the use of a dialogue object causes problems.

Presentation aspects. Both data and actions—as long as they are not hidden—must be made available to the user. *Presentation rules* are generally defined globally for the overall application—independently from individual dialogue types. Such rules determine, e.g., graphical widgets to be used for visualisation (buttons or menus for actions, text fields for textual data, etc.). Furthermore, they can control the layout of visualisations based, e.g., on the space available on the screen and on the information to be conveyed.

D-types will, however, provide parameters for such presentation rules. Parameters are, e.g., *labels* that are to be used for visualising dialogue objects

(title bar) and individual data or actions. Other parameters are diverse *semantical properties* of data such as *emphasis*, *priority*, *adhesion*, etc. [8]. If statistical anomalies are discovered during the execution of a dialogue object (a shop whose sales significantly exceed the average) *local presentation rules* can, e.g., characterise such anomalies as important. Global presentation rules will then use the parameter *important* to choose red colour, flashing font, etc.

Conceptually, presentation design of d-types is therefore concerned with a description of the data and actions involved and/or with the specification of rules that automatically compute such descriptions, rather than with an explicit assignment of presentations. We specify labels, semantical properties, adhesions between data items, rules, etc. but no concrete widgets, colours or fonts.

At the implementational level, presentation rules use these descriptions to create physical presentations of dialogue objects. There can be several presentation rules available for one concept which can, e.g., create a graphical or a forms-based presentation for the same d-type. Presentation rules are selected at system run-time in dependence of the user's preferences or actions invoked.

3.3 OLAP Actions

In OLAP applications, roll-up operations and drill-down operations are used for generalisation and specialisation of fact tables. Aggregation of detailed data to create summary data is called roll-up. Usually, roll-up is based on two operations: grouping of data according to some characteristics (e.g., total sales by city and product) and navigation through an attribute hierarchy (e.g., from sales by city towards sales by state and sales by country). Navigation to detailed data from summaries is called drill-down. It provides the data set that was aggregated (e.g., displaying the 'base' data for total sales figure for the state CA). Selection which is called slicing in OLAP applications defines a subdatabase (e.g., sales where city="Berkeley" or reducing the relationship dimensions by specifying coordinates of remaining dimensions). The last observation shows that minimum and maximum functions yield the same result after application of roll-up operations or drill-down operations. The same property is valid for pull, pivoting and push operations [1]. These functions can be generalised to reordering or rearrangement functions. OLAP operations can be completely based on operations defined for ER models [18].

- Classical database operations are also applicable to views. For this reason, the following functions are defined for views:
 - Selection on relationship types is the ER expression for *dice*.
 - The *slice* operation is expressed by projection on relationship types.
 - The set-theoretic operations *union*, *intersection*, *cartesian product* and *difference* and component *renaming* are elements of the ER algebra.
- *Calculations* within one component type or across component types are expressed by algebraic functions. *Ranking* functions are based on the computation of sets, ordering and creating supporting views. Extensions such as *tertile*, *quartiles*, *ratio_to_report*, *cume*, *moving-average*, *moving-sum* are expressible in ER-SQL [18].

- Visualisation functions do not produce another view. They are used for re-ordering the schema. Functions such as *nest* and *rotate* can be represented by ER operations. *Ranking* functions are expressed by **order by** constructs. Dynamic breakpoints are used to start a new view for computation. They require the utilisation of dynamic SQL. Extending ER-SQL to dynamic ER-SQL can be performed in the fashion known for SQL.
- Aggregation can be expressed by view definition in database languages.
 - The *roll-up* function (also called drill-up) is used for dimension reduction on the basis of aggregation functions. Navigation through an attribute hierarchy can be expressed by escorting queries. The complex *cube* operation is a set of roll-up operations and expressible in ER-SQL.
 - The *drill-down* function is just the inverse operation of roll-up.
 - The aggregation functions *min*, *max*, *sum*, *avg*, *count* can be expressed by summarisation. This function generalises the aggregation functions.
- Schema restructuring has been generalised by Gyssens and Lakshmanan [5]. *Classification*, *fold* and *unfold* can also be generalised in the ER algebra. *Unfold* defines a new view on a relationship type and a set of components of the relationship type by introducing a new type on the set and a new relationship type with the remaining components of the first type and the set component. *Unfold* generalises the *unnest* operation. *Fold* restructures a schema to a new schema for a component of a chosen relationship type. *Fold* generalises the *nesting* operation. *Classification* is a specific grouping operation. Schema restructuring operations are operations which can be expressed by graph grammar rules.
- OLAP applications are also based on analytical functions which are defined on the basis of mathematical models. Since OLAP data can be understood as derived data which can be materialised, analytical functions can also be defined for databases.

From the user's point of view typical actions invoked—besides choosing a completely new view in the sense presented above—are devoted to achieve finer or coarser tuning of the presented information:

- A user may choose to extend or restrict the fact list. In the example sketched in Section 3.2, the facts **Quantity** and **Profit** may be added. Note that the evaluation of the corresponding query may exploit data already presented as an intermediate result.
- A user may choose to add or remove a dimension. For example, a manager looking at the data selection indicated above may remove the product dimension and simply look at **Money_sales** in different regions. Again, query evaluation is simplified by the existing data contents of the dialogue object.
- A user may add or remove dimension attributes. The required processing is analogous to changes in the fact list.
- The user may change dimension granularity. Such drill-down or roll-up operations switch to a smaller or larger dimension. For example, the user may replace **REGION** by **SHOP** or **PRODUCT** by **BRAND** walking along the hierarchies in the snowflake schema.

- The selection criteria may be weakened or strengthened leading to a larger or smaller result. Here again, the reuse of the existing query result may help to fasten query processing.
- The user may mark a section of the presented data and invoke an aggregate function on them such as summation, average computation, normalising at a given index 100, etc. In our grocery store example the money sales per shop could be considered.
- Grouping, ordering and presentation style may be changed. As already stated above, this does not lead to a new query and only affects general presentational issues. It is possible to achieve this functionality by using different presentation rules.

4 Conclusion

In this paper we argued that a conceptual view of data warehouses and management support information systems with OLAP functionality can be achieved by exploiting the idea of dialogue objects which was coined in the context of dialogue information systems supporting high-skilled clerical work. Firstly, the warehouse schemata as well as the data presented to managers can be derived as views on the operative database(s). Secondly, the functions required in OLAP applications can be coupled with these views.

We remark that the idea of extended views has also been fruitfully expanded and applied in theory and practice to support the design of database-backed information services on the WWW [2]. The fact that users of information services are by no means better trained in computer usage than managers underlines the appropriateness of the suggested approach.

How to realise the underlying views to enable fast access is left as a technical question of which views or intermediate views should be materialised or not. Currently, it is assumed that the data warehouse itself is materialised as a first large view but views on the warehouse are not. We would like to emphasise that this is only one possible solution.

Similarly, from dialogue objects we inherit the methodological separation of data contents on the one side and its presentation on the other. Hence, our suggested approach allows a threefold split in the design of OLAP applications:

- a conceptual task oriented towards the design of dialogue structures, i.e., views, visibility of views, actions and availability of actions;
- a more technical task to define suitable view realisation strategies;
- a general design task to specify presentation rules for the creation of suitable screen representations for view data and actions.

Naturally, the individual design tasks can be performed by different designers being experts in the respective domain, i.e., application design, database management and interface design.

References

1. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional database. In *Proc. Data Engineering Conference, Birmingham*, pages 232–243, 1997. 365
2. T. Feyer, K.-D. Schewe, and B. Thalheim. Conceptual design and development of information services. In T.W. Ling, S. Ram, and M.L. Lee, editors, *Conceptual Modeling – ER '98*, LNCS 1507, pages 7–20. Springer, 1998. 358, 361, 367
3. C. Floyd, F.-M. Reisin, and G. Schmidt. STEPS to software development with users. In C. Ghezzi and J.A. McDermid, editors, *ESEC'89*, LNCS 387, pages 48–64. Springer, Berlin, 1989. 354
4. P. Gluchowski. Data warehouse. *Informatik Spektrum*, 20(1):48–49, 1997. 355, 355
5. M. Gyssens and L.V.S. Lakshmanan. A foundation for multidimensional databases. In *Proc. 22nd VLDB Conference, Mumbai (Bombay), India*, 1996. 366
6. B. Jahnke, H.-D. Groffmann, and S. Kruppa. On-line analytical processing (OLAP). *Wirtschaftsinformatik*, 38(3):321–324, 1996. 355
7. R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, 1996. 355, 355, 356, 356, 356, 356
8. J. Lewerenz. On the use of natural language concepts for the conceptual modeling of interaction in information systems. In *Proc. of NLDB'99, Klagenfurt*, 1999. 365
9. A. Motro. Superviews: Virtual integration of multiple databases. *IEEE ToSE*, 13(7), 1987. 355
10. H. Mucksch, J. Holthuis, and M. Reiser. Das Data-Warehouse-Konzept – Ein Überblick. *Wirtschaftsinformatik*, 38(4):421–433, 1996. 355, 355, 356
11. S.B. Navathe, R. Elmasri, and J.A. Larson. Integrating user views in database design. *IEEE Computer*, 19(1):50–62, 1986. 355
12. N. Pendse. The OLAP Report. Available through <http://www.olapreport.com/>. 355
13. B. Schewe. *Kooperative Softwareentwicklung – Ein objektorientierter Ansatz*. Deutscher Universitätsverlag, Leverkusen, 1996. 356, 361
14. B. Schewe and K.-D. Schewe. A user-centered method for the development of data-intensive dialogue systems – an object oriented approach. In E.D. Falkenberg, W. Hesse, and A. Olivé, editors, *Information System Concepts*, pages 88–103. Chapman & Hall, 1995. 354, 361
15. K.-D. Schewe and B. Schewe. View centered conceptual modelling – an object oriented approach. In B. Thalheim, editor, *Conceptual Modeling – ER '96*, LNCS 1157, pages 357–371. Springer, 1996. 356, 361
16. K.-D. Schewe and B. Schewe. Integrating database and dialogue design. *Knowledge and Information Systems*, 1999. To appear. 363
17. B. Thalheim. Foundations of entity-relationship modeling. *Annals of Mathematics and Artificial Intelligence*, 7:197–256, 1993. 355, 355, 355
18. B. Thalheim. *Fundamentals of entity-relationship modeling*. Springer, Heidelberg, 1999. 356, 357, 357, 365, 365
19. E. Thomson. *OLAP Solutions: Building Multidimensional Information Systems*. John Wiley & Sons, New York, 1997. 355

Resolving the “Weak Status” of Weak Entity Types in Entity Relationship Schemas

Mira Balaban and Peretz Shoval

Information Systems Program
Dept. of Industrial Engineering and Management
Ben-Gurion University of the Negev, P.O.B. 653, Beer-Sheva 84105, Israel
`mira@cs.bgu.ac.il`
`shoval@bgumail.bgu.ac.il`

Abstract. Entity Relationship schemas include *weak* entity types, whose entities are identified by their inter-relationships to other entities. During the translation of the EER schema into a logical database schema, the weak entity types are either translated into logical units of the database (a relation or a class), or are embedded as attributes of other logical units. Formal studies of EER schemas either ignore the presence of weak entity types, or simplify their dependency structure. The presence of weak entity types in an EER schema may be problematic: A weak entity may not be easy to identify because it may be related to other weak entities in various ways, thus causing problems in schema comprehension, as well as in mapping it to a logical database schema.

We claim that the presence of weak entity types in an EER schema belongs to an intermediate design stage, but the final EER schema must not include such entity types. We introduce an algorithm for resolving the status of weak entity types, following user directions. If the directions are legal, the algorithm yields an EER schema without weak entity types. The advantage of our approach is twofold: First, the translation of an EER schema into a logical database schema can be fully automated. This is essential for upgrading the EER model to support full database management. Second, it enables a fully formal study of the EER model.

1 Introduction

ER is a popular visual data model for describing entities, their properties, and inter-relationships. A set of entities that share a common structure is captured as an *Entity Type*. Regular properties of entities are captured as their *Attributes*. Interactions among entities are modeled by *Relationship Types*. Cardinality constraints are set on relationship types, and characterize numerical dependencies among entities within the relationship types. The model was introduced by Chen [5], and received many extensions and variations, which are generally termed the Enhanced ER (EER) model. Traditionally, EER schemas are used for conceptual database design, and are translated into logical database schemas, usually relational or object-oriented (OO) schemas [6]. Application

program transactions and consistency checks are added directly to the target database schema.

The EER model is value oriented, in the sense that it assumes that entities in a given entity type are uniquely identified by an attribute (or a set of attributes) of that entity type, called a *key*. Under this assumption, an entity type must have a key, since this is the only means for referencing its entities. However, this assumption is too restrictive since in some entity types entities are identified by their inter-relationships to other entities. For example, in a medical clinic (see Figure 1), a visit is identified by its date, and by the patient entity that participates in the visit. Such entity types (as **Visit**) are distinguished in EER schemas as *weak* entity types. Their entities are related through *identifying relationship type(s)* to their *owner* entity(s). They are identified by their owner entities, and by their partial key attributes, if any. During the translation of the EER schema into a logical database schema, the weak entity types are either translated into logical units of the database (a relation or a class) with references to the translations of their owner entity types, or are embedded as attributes of other logical units.

The semantics and manipulation of weak entity types may become complicated, when a weak entity type has multiple owners, or when it has relationships with other entity types, or when the owners are, themselves, weak. In the latter case, the embedding of a weak entity type as an attribute of the translation of another entity type might be indirect, since the owner entity type might itself be embedded as an attribute. Resolving the embedding in an algorithmic way requires an inductive construction of the *embedding entity type* for a weak entity type. When a weak entity type E_1 is translated into a logical unit of the database, while its weak owner E_2 is resolved to be embedded as an attribute of its own (possibly indirect) owner E_3 , the translation of E_1 must somehow know to reference E_3 .

In general, EER models do not restrict the “ownership structures” of weak entity types. As a result, an EER schema can include complicated *ownership paths*, from a weak entity type to its owners. For example, there can be multiple identification paths between a weak entity type to one of its owners (possibly indirect), or there can be cycles of ownership relationships, etc. The understanding of such structures becomes hard, the unclear semantics turn the schemas incoherent, formal tools that test the consistency and correctness of schemas do not apply to the general case, and mapping algorithms from an EER schema to other schemas do not apply to the general case as well. Consequently, in practice, the embedding of weak entity types is, usually, done manually by the modeler carrying the EER schema translation. Standard translation algorithms fall short of full algorithmic translation [6]. Moreover, formal studies of EER schemas either ignore the presence of weak entity types [8,9,4], or simplify their dependency structure [7].

The presence of weak entity types seems important for representing and comprehending reality in the initial stages of design. But in the final stage of design, it is advisable, because of the above mentioned problems, that weak

entity types are transformed either into strong entity types, or into attributes of other entity types. Moreover, conceptual data models have become significant in heterogeneous environments, where they are used as unifying meta-schemas, that coordinate the cooperation of system modules that assume different data models. This status requires that they are formally defined. In particular, the vague status of weak entity types prevents the upgrading of the EER data model into a conceptual DBMS¹.

We claim that although the presence of weak entity types was practically shown essential for data analysis and design, their status is not well defined. Moreover, they are treated differently in the various ER models. The goal of this work is to clarify the problematics of weak entity types, and suggest a solution that recognizes their importance in the early stages of a system design. We conceive conceptual modeling as a process of construction of conceptual schemas. The process starts with a loosely defined schema, and proceeds towards a well defined one. The elimination of weak entity types, which is the focus of this paper, is part of this process. The final schema includes no weak entity types, and is, semantically, well defined. For that purpose, we introduce an algorithm for resolving the status of weak entity types in EER schemas, within the context of the schema. The algorithm is, in essence, an upgrading of the standard handling of weak entity types to the EER schema level. The user marks the weak entity types that are candidates for being embedded as attributes of other entity types. If the embeddings are possible (legal), they are built by the algorithm. Otherwise, the algorithm fails, pointing out the cause of failure. Overall, if the algorithm succeeds, it yields a new EER schema without weak entity types, and with *reference attributes* instead.

The contribution of this paper is in resolving the status of weak entity types in EER schemas. The algorithm we introduce can be applied as the last step in the design of the schema. Once this is done, all existing formal investigations of EER schemas are applicable, and EER schemas can be formally defined as an abstract conceptual level on top of a logical database schema. That is, the resulting EER schema can be automatically implemented in terms of a logical database schema. We believe that our approach clearly separates the conceptual modeling stage, where the user must be consulted, from the implementation stage, that should be carried out automatically.

In Section 2 we discuss the issue of weak entity types in EER schemas, emphasizing complex interrelationships between a weak entity type to its owners. In Section 3 the resolution algorithm for weak entity types is introduced and demonstrated. Section 4 is the conclusion.

¹ Indeed, the motivation for this work came from the need to implement a MEER schema (EER with structure Methods [3,2]) in terms of an object-oriented database schema. The complexity of ownership structures for weak entity types made the translation impractical.

2 Weak Entity Types in the EER Data Model

An EER schema consists of *Entity Type* symbols, *Relationship Type* symbols (each with associated arity), *Role Name* symbols, and *Attribute* symbols. Entity type symbols can be *strong* (denoting non-weak entity types) or *weak*. Attribute symbols can be *simple* or *composite*. A composite attribute symbol is associated with a set of attribute symbols (other than itself). Furthermore, each attribute symbol is either *single-valued* or *multi-valued*.

A key of a type is a means for identifying the instances of the type via their attributes. A key of a strong entity type symbol and of a relationship type (if any) is an attribute of the type. A key of a weak entity type symbol is defined through related *owner* entity type symbols. Every weak entity type symbol E is associated with binary relationship type symbols that are singled out as the *identifying relationship types* of E . The entity type symbols associated with E through these relationship type symbols are termed the *owner entity types* of E . The cardinality constraints associated with the identifying relationship types of E guarantee that every entity e of E is related through an identifying relationship type to a single owner entity. Moreover, e can be identified by its owner entities and its values for the partial key attributes of E . That is, the key of E consists of the keys of its owners, and its own *partial key* (if exists), which is any of its attributes.

For the sake of keeping this paper short, we avoid presenting a formal definition of the EER data model, its semantics and consistency. Such definitions have been provided elsewhere [10,4,1]. Figure 1 presents an EER diagram for a medical clinic. Rectangles describe entity types, diamonds describe relationship types, circles describe attributes. Solid lines among rectangles describe entity type hierarchies, with arrow heads pointing to super entity types, and dotted line rectangles and diamonds stand for weak entity types and their identifying relationship types (with arrow heads pointing to owner entity types).

In Figure 1 we see that **Schedule** is a weak entity type owned by the strong entity type **Physician**. A **Schedule** entity is identified by its physician's license-number and a day, which is its partial key. A more complex ownership structure is demonstrated by the weak entity type **Lab-test-prescribed**, which is owned by the **Lab-test** and the **Visit** entity types. **Visit** is itself weak; it is owned by **Patient**, and identified by the **Patient** ID, and the **Visit** date. The **Visit** entity type stands in a regular relationship with **Physician**. If **Visit** is resolved to be embedded as an attribute of its owner **Patient**, then whoever references **Visit**, needs to know its **target_entity_type**, i.e., **Patient**. This applies to the **treating** and the **prescription** relationship types. Hence, every embedding requires the specification of the final **target_entity_type**. In general, the presence of complex owner dependencies complicates the formal treatment. Clustering algorithms that instruct the user to embed weak entity types within their owners, fall short of handling multiple owners.

An EER schema that includes weak entity types cannot be fully formalized, since the user must provide information about the intended status of weak entity types, i.e., whether a weak entity type symbol stands for a real entity type or for

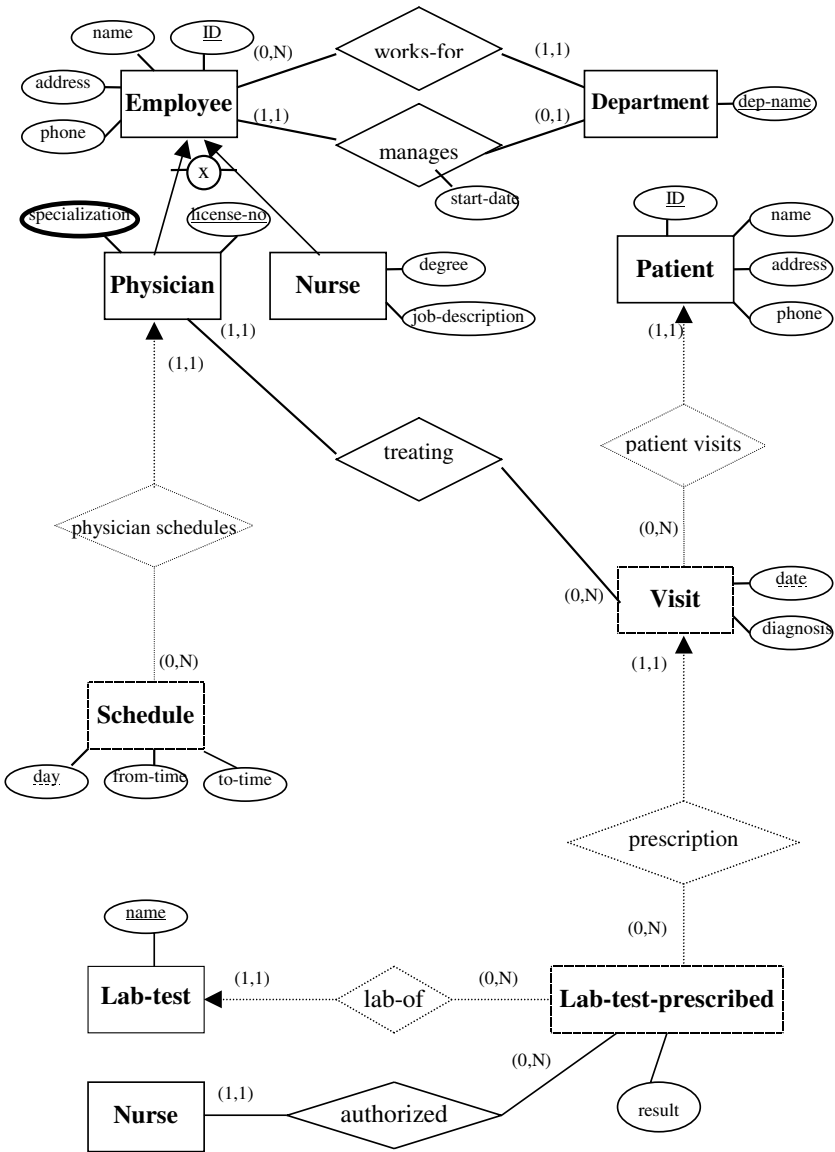


Fig. 1. An EER diagram for a medical clinic information system

an attribute of another entity type. This is a conceptual decision, and should be left to the modeler. In particular, attribute embedding implies that conceptually, references to entities of the embedded weak entity type can be replaced by references to the entities of its **target_entity_type**. When the missing information is provided it can still be illegal since it might yield an illegal schema, or an inconsistent one.

3 Resolving the Status of Weak Entity Types

Weak entity type symbols in the schema are resolved either to stay as entity type symbols, or to turn into attribute symbols associated with other entity types. In the first case, the owner entity type symbols of the weak entity type symbol E are combined to form a new composite *reference attribute* of E . In the second case, the weak entity type symbol becomes an attribute of its “nearest” (direct or indirect) owner entity type symbol that is resolved as an entity type. In order to determine this closest entity type symbol, an inductive construction of the “target” entity type symbol is required.

The algorithm for removing the weak entity type symbols from an EER schema by resolving their final status works in four stages: In the first stage, it is verified that the schema does not include ownership cycles as in Figure 2. Such cycles are illegal since they imply infinite keys². In the second stage, the user marks, following some guidelines and constraints, the final status of the weak entity type symbols. In the third stage, the “target” entity type symbol for weak entity types that have an “attribute” destination is determined. In the fourth stage, the new EER schema, that includes no weak entity type symbol, is generated. The algorithm is introduced below, and its application to the medical clinic example from Figure 1 is demonstrated.

Stage I: Detect ownership cycles: If a cycle is detected, output the entity type symbols in the cycle and stop.

Stage II: Determining the final status of the weak entity type symbols:

1. A weak entity type symbol with multiple owners is resolved to stay as an entity type (note that a single entity type symbol can play the role of multiple owners, through several identifying relationship types). The rationale here is to prevent replication of the weak entity type as attributes of all of its owners. For example, in Figure 1, **Lab-test-prescribed** is resolved to stay as a regular entity type because it is owned by **Visit** and **Lab-test**.
2. A weak entity type symbol E with a single owner, such that E is not associated with a non-identifying relationship type symbol, and is not the owner

² Later on we suggest to relax the notion of a key of a weak entity type to include reference attributes. In that case, ownership cycles are legal, provided that not all entity types in an ownership cycle are resolved to be embedded as attributes of their owner entity types.

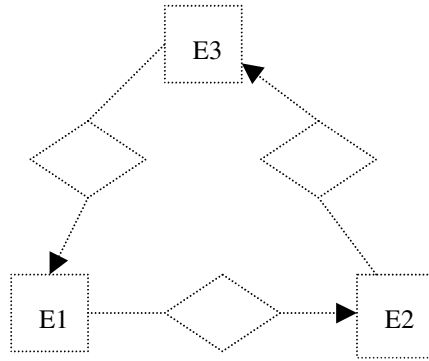


Fig. 2. A cycle of owner entity type symbols

of another entity type symbol, is resolved to be embedded as an attribute of its owner entity type symbol. For example, in Figure 1, **Schedule** is resolved to be embedded as an attribute of **Physician**.

3. Otherwise, the final status is determined by the user (modeler). That is, a weak entity type symbol that has a single owner, and is either associated with a non-identifying relationship type symbol, or is the owner of another weak entity type, is resolved either to stay as an entity type, or to be embedded as an attribute of its owner entity type symbol. For example, **Visit** in Figure 1, might have either an “attribute” or an “entity type” resolution.

If the weak entity type is constrained by a non-trivial (not (o, N)) cardinality constraint, it is advised to resolve it to stay as an entity type, since attribute embedding requires composition of the cardinality constraints along the embedding path. A composition of that kind may be not informative for non-trivial constraints. For example, the composition of $(3, 5)$ with $(4, 10)$ yields $(12, 500)$.

Stage III: Determining the “target” entity type symbol for weak entity types that have an “attribute” destination:

The **target_entity_type** of E is the weak entity type symbol in which E will be embedded as an attribute. It is needed for the case where E is the owner of another weak entity type symbol E^1 , that is resolved to stay as an entity type, or if E is related with a non-identifying relationship type symbol R . This situation is demonstrated in Figure 3. The entity type symbol E^1 needs a *reference attribute* to E , as part of its key, and R needs to relate E , with other entity type symbols. Since E is resolved to turn into an attribute, E^1 and R need a reference to the entity type symbol in which E is embedded. This is the **target_entity_type** of E . For example, in Figure 1, if **Visit** is resolved to be embedded as an attribute of **Patient**, then **treating**, needs to reference **Patient** as the **target_entity_type** of **Visit**. Similarly, **Lab-test-prescribed** which has an “entity type” resolution, needs **Patient** as a reference attribute.

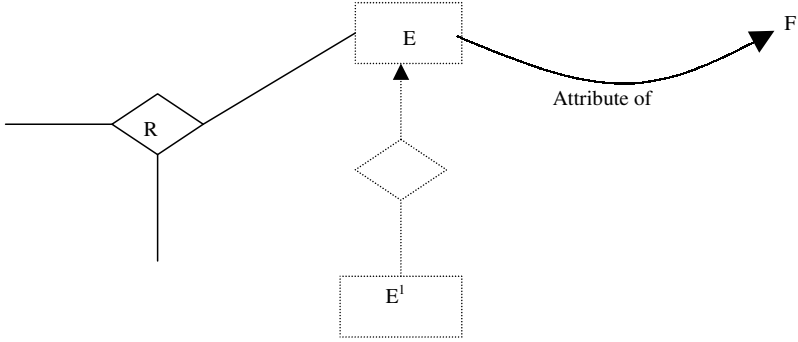


Fig. 3. A weak entity type symbol with an “attribute” labelling, that is the owner of another weak entity type symbol, and is related through a non-identifying relationship symbol

1. For every entity type symbol E that is either strong or is resolved to stay as an entity type symbol, **target_entity_type**(E) = E .
2. For every entity type symbol E that is resolved to be embedded as an attribute, and is owned by the (single) entity type symbol E' :
target_entity_type(E) = **target_entity_type**(E').

The **target_entity_type** labelling of entity type symbols in a schema can be computed inductively, starting from the strong entity type symbols, and the weak entity type symbols that have an entity type resolution. At the end of this computation, all entity type symbols have a **target_entity_type** labelling, since the schema does not include ownership cycles.

The computation of the target entity type of an entity type symbol E in a schema, is associated with the cardinality constraints between E to its target entity type. This computation is obtained by composing the cardinality constraints along the role path between E to **target_entity_type**(E). For the sake of simplicity we avoid specifying this computation here.

Stage IV: Constructing a new EER schema $\mathcal{ER}'$ that does not include weak entity type symbols, from a given schema $\mathcal{ER}$:

Insert to $\mathcal{ER}'$ all elements of $\mathcal{ER}$.

1. **Weak entity type symbols that have an “entity type” resolution:**
 For every such entity type symbol E that is owned by the entity type symbols $E_1, \dots, E_k$, with **target_entity_type**(E_i) = E_i^1 , $i = 1, k$, add to E a composite attribute named *owner* which is associated with the *reference attributes* E_i^1 , $i = 1, k$. Add *owner* to all keys of E . This situation is demonstrated in Figure 4. The newly added reference attributes are denoted with bold dashed circles. In Figure 7, the assumption is that the weak entity type symbol **Visit** is resolved to be embedded as an attribute of **Patient**. Hence,

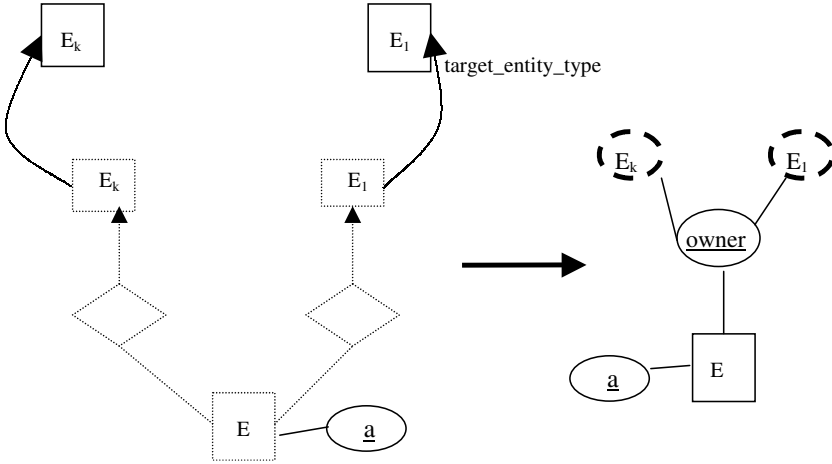


Fig. 4. Reference attributes added to the key of a weak entity type symbol with an “entity type” labelling

Lab-test-prescribed has two reference attributes: **Lab-test** and **Patient**.

For each identifying relationship between E to some E_i it is up to the user to decide whether it should be removed or not. Removing such a relationship implies that the user thinks that there is no need to relate entities of E_i^1 to entities of E . The other direction is already covered by the newly added reference attributes of E . In any case, a relationship that is resolved to stay in $\mathcal{ER}'$ is not identifying any more. In the medical clinic example of Figure 1, the decision was not to remove such relationships. Hence, in Figure 7 the relationships **Lab-of** and **prescription** stay as regular relationships.

2. **Non-identifying relationship type symbols:** For every relationship type symbol R that relates in $\mathcal{ER}$ the entity type symbols $E_1, \dots, E_k$, replace them in $\mathcal{ER}'$ with **target_entity_type**(E_i) = E_i^1 , $i = 1, k$, respectively. Whenever E_i^1 is different from E_i , the cardinality constraint on its role in R is set to $(0, N)$. This situation is demonstrated in Figure 5. In the medical clinic example, the **treating** relationship between **Physician** and **Visit** relates, in the transformed schema in Figure 7, **Physician** with **Patient**.

After this stage, non-identifying relationship type symbols in $\mathcal{ER}'$ relate only entity type symbols with “entity type” resolution.

3. **Weak entity type symbols that have an “attribute” resolution:** While there exists in $\mathcal{ER}'$ an entity type symbol E with “attribute” resolution, which is not the owner of another (weak) entity type symbol with an attribute resolution, repeat the following:
 - Assume that the owner of E in $\mathcal{ER}'$ is E^1 , via the identifying relationship R , and with role names RN and RN^1 , respectively. Assume that E has the attributes $A_1, \dots, A_n$. Then, add to E^1 a composite attribute E ,

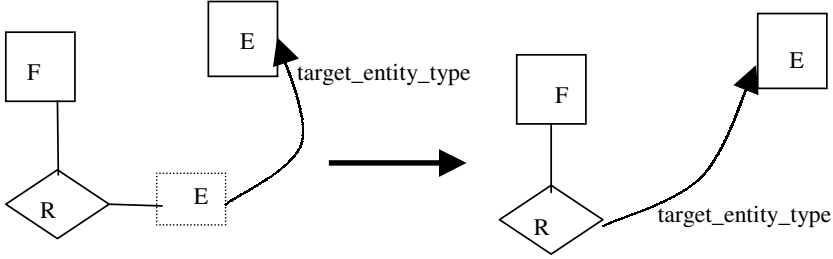


Fig. 5. Transforming the reference in non-identifying relationship type symbols

that is associated with the attributes $A_1\_of\_E$, $\dots$, $A_n\_of\_E$. If the cardinality constraint imposed on the role name RN is $(1,1)$, i.e., at most a single E entity is identified by a given E^1 entity, then the attribute E of E^1 is marked as single-valued. Otherwise, it is multi-valued. This situation is demonstrated in Figure 6.

- Remove E and its identifying relationship from $\mathcal{ER}'$.

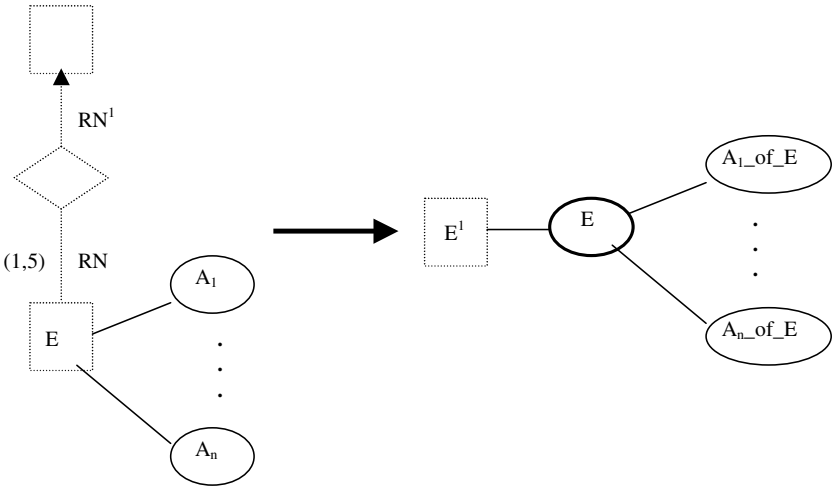


Fig. 6. Embedding a weak entity type with an “attribute” resolution as an attribute of its owner

A possible schema that results from the application of this procedure to Figure 1 is described in Figure 7. In the new schema **Schedule** turns into a multi-valued attribute of **Physician** since it has a mandatory attribute labelling, according to

Stage II. **Lab-test-prescribed** stays as an entity type symbol since it has two owners in the original schema. It has two reference attributes, **Lab-test** and **Patient**, which are the **target_entity_types** of **Lab-test** and **Visit**, respectively. The decision to turn **Visit** into an attribute of its **target_entity_type Patient**, was taken mainly in order to demonstrate the case of “attribute” labelling for a weak entity type symbol that is related through a regular relationship type, and is also the owner of another weak entity type symbol with an “entity type” labelling.

Figure 8 describes another possible result of Figure 1, where **Visit** is resolved to stay as an entity type. From a conceptual viewpoint, Figure 8 is preferable.

3.1 Modification of the EER Data-Model Definition

Two modifications are needed in the schema definition and its semantics:

1. **Reference attributes:** This is a special kind of attribute symbols, that take the same name as an entity type symbol in the schema. The type of a reference attribute symbol E of a type symbol T must be included in the domain of entities. If $dom(E, T)$ denotes the type assigned to the attribute E of T , and D is the domain of entities in a database instance of the schema, then $dom(E, T) \subseteq D$.
2. **Consistency:** Additional condition for the consistency of a database instance $\mathcal{D}$ of a schema:
For every reference attribute symbol E of a type symbol T , its type is exactly the extent of E in $\mathcal{D}$. That is, $dom(E, T) = E^{\mathcal{D}}$.

3.2 Allowing Ownership Cycles

The extensional (attribute) meaning of a key does not allow for ownership cycles, since they imply that the weak entity types in a cycle have infinite keys. However, once the EER model is extended with reference attributes, they can be part of a key. The implication of this extension is that the original EER schema can allow for ownership cycles, provided that not all weak entity type symbols in a cycle are resolved to be embedded as attributes in the revised schema (assigned an “attribute” labelling in Stage II of the algorithm). Figure 9 demonstrates such an illegal labelling. The problem with cyclic “attribute” labellings is that the entity type symbols in such a cycle do not have a **target_entity_type** where they should be embedded as attributes. Illegal cyclic labellings are detected in Stage III of the algorithm, since they leave entity type symbols without **target_entity_type** labellings (the weak entity type symbols in a cycle of attribute labellings cannot be reached by the inductive construction). In that case, the algorithm can either output the entity type symbols in an illegal ownership cycle and terminate, or modify stage II.

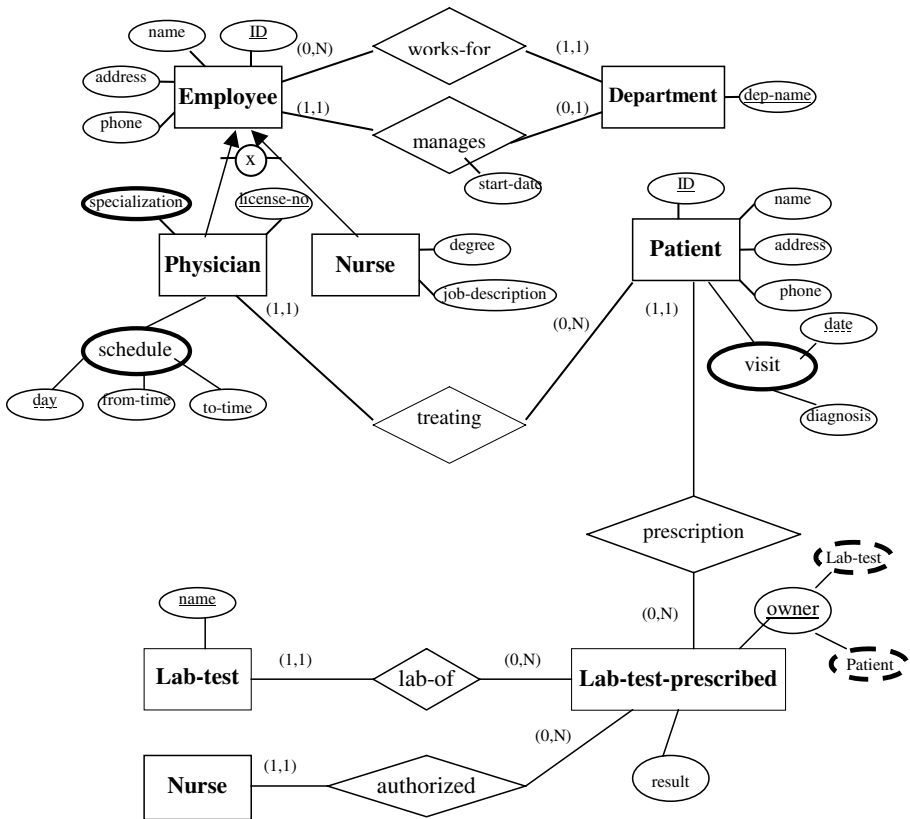


Fig. 7. A Weak-entity-type-symbols less EER diagram for the medical clinic information system

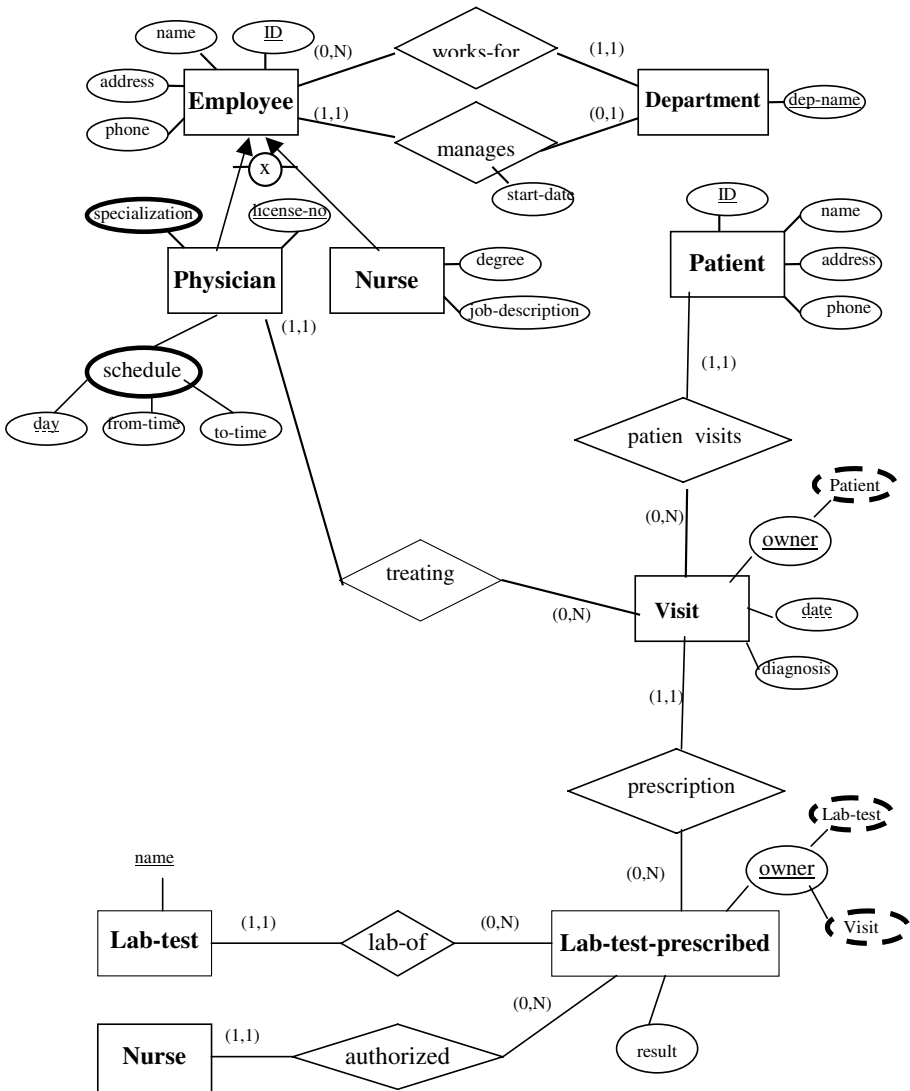


Fig. 8. A conceptually preferable weak-entity-type-symbols less EER diagram for the medical clinic information system

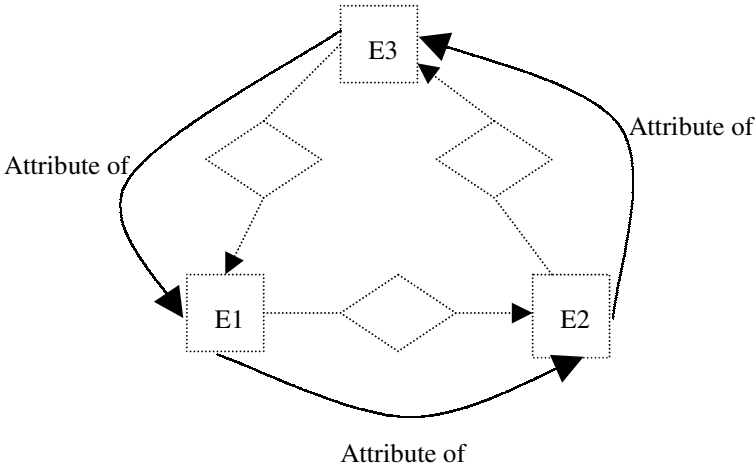


Fig. 9. A cycle of owner entity type symbols with illegal “attribute” labelling

4 Conclusion

This work is part of our more general work on formalizing the EER model for database programming level. We encountered problems with handling weak entity type symbols in two cases. The first is in our work on the MEER model, which is an EER schema that is enhanced with structure methods that are cardinality sensitive [3]. In order to implement the MEER model on top of a database schema, there is a need to implement all the embeddings of weak entity type symbols, and to develop the machinery to reference these embeddings. This process becomes unreasonably complex, and calls for separation between the status resolution of the weak entity type symbols, to the implementation of the methods. Once the two processes are separated, it makes sense to abstract the weak entities part on the EER conceptual level. This way we separate this process from the implementing database model. The other case is in our work on abstraction, hierarchy and clustering in EER diagrams. Most suggestions try to cluster weak entities with their owners. But this process is not well-defined due to the complex structure of weak-owner dependencies, as described in this paper.

In this paper we introduced an algorithm for resolving the status of weak entity types in EER schemas. If the algorithm succeeds, it yields an EER schema without weak entity types. The contribution of this paper is in resolving the “weak” status of weak entity types in EER schemas. The algorithm we introduce can be applied as the last step in the design of the schema. Once this is done, all existing formal investigations of EER schemas are applicable, and EER schemas can be formally defined as an abstract conceptual level on top of a logical database schema.

In the Object-Oriented (OO) data model there are no “weak classes”, since there is no mandatory key requirement. Yet, the conceptual role played by weak entity types seems useful, since it enables to mark classes that might later on be embedded as attributes of other classes. The algorithm for resolving the status of weak entity types applies in this case as well.

References

1. M. Balaban and P. Shoval. Enhancing the ER model with structure methods. *Journal of Database Management*, 10(4), 1999. 372
2. M. Balaban and P. Shoval. EER as an active conceptual schema on top of a database schema – object-oriented as an example. Technical report, Information Systems Program, Department of industrial Engineering and Management, Ben-Gurion University of the Negev, ISRAEL, March 1999. 371
3. M. Balaban and P. Shoval. MEER – an EER model enhanced with structure methods. *submitted.*, 1999. 371, 382
4. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer Academic Publishers, 1998. 370, 372
5. P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. 369
6. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, 1994. 369, 370
7. P. Jaeschke, A. Oberweis, and W. Stucky. Extending er model clustering by relationship clustering. In *Entity-Relationship Approach*, pages 451–462. North-Holland, 1993. 370
8. M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990. 370
9. B. Thalheim. Fundamentals of cardinality constraints. In *Entity-Relationship Approach*, pages 7–23. North-Holland, 1992. 370
10. B. Thalheim. *Fundamentals of Entity-Relationship Modeling*. Springer-Verlag, 1998. 372

A Taxonomy of Recursive Relationships and their Structural Validity in ER Modeling

James Dullea¹ and Il-Yeol Song²

¹Boeing Phantom Works, The Boeing Company, Philadelphia, PA 19142
Email: james.dullea@boeing.com

²College of Info. Science and Technology, Drexel University, Philadelphia, PA 19104
Email: song@drexel.edu

Abstract. In this paper, we present the complete classification of recursive relationships and the criteria that contribute to the structural validity of modeling recursive relationships within the entity-relationship (ER) diagram. Unlike typical other analyses that use only maximum cardinality constraints, we have used both maximum and minimum cardinality constraints in defining the properties and their structural validity criteria. We used the notions of role uniqueness, path connectivity, and cardinality constraints to derive a complete and comprehensive set of decision rules. Five rules and three corollaries were established to determine structural validity of recursive relationships. The contribution of this paper is to present a complete taxonomy of recursive relationships with their properties as well as the decision rules for their structural validity. These decision rules can be readily applied to real world data models regardless of their complexity. The rules can easily be incorporated into the database modeling and designing process, or extended into case tool implementations.

1. Introduction

Entity-relationship (ER) modeling [4] is the foundation of various analysis and design methodologies for the development of information systems and relational databases. A key measure of success in the design of these models is the level that they accurately reflect the real world environment. A model can be a very complex abstract structure, and designers are highly prone to making many mistakes that incorporate inconsistencies into the structure. There is various supporting empirical evidence [2, 5, 9] that concludes errors, mistakes, or inconsistencies made in the early stages of the software development life cycle are very expensive to correct. Boehm [3] states that the cost difference to correct an error in the early phases as opposed to the post-implementation phase is on the order of a ratio of one to one hundred. Left undetected these inconsistencies become very costly to correct, so early discovery of an error is highly desirable.

The structural validity of an ER diagram is concerned whether or not a given ER diagram contains any constructs that are contradictory each other. An ER diagram with at least one inconsistent cardinality constraint is structurally invalid. Deciding whether a particular ERD is valid or not is sometimes a difficult issue to many database designers. For example, how do we know whether the diagram shown in Figure 1 is valid? In this paper, we present decision rules whether a particular recursive

relationship, also referred to as a recursive relationship, is valid or not. We explain why Figure 1 is invalid in Section 4 using our decision rules.

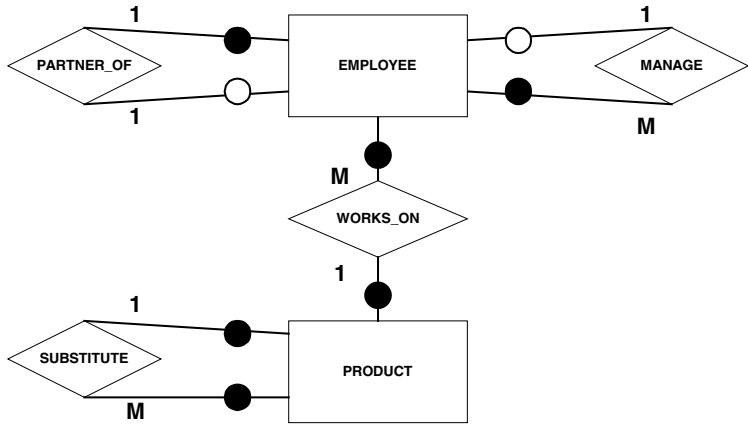


Figure 1: An ERD with Invalid Recursive Relationships

Direct treatment of structural validity as a concept is rare in the literature. Some structural validity rules are partly discussed in other publications, such as [15, 1, 8, and 14], but they don't exhaustively cover all the combinations of maximum and minimum cardinality. Indirect treatment of the subject materials are usually confined to making buttress points concerning other issues. Structural validity in the literature has almost always been treated as a property of a study; in this paper we treat it as the object of our study. In our study we evaluate each relationship type as part of the overall diagram and address how they coexist with other relationship types within the model.

The structural validity of ternary relationships has been addressed in [11, 7]. In this paper we focus on recursive relationships. In the literature, only two types of recursive relationships - symmetric and asymmetric [15, 6] and their validity [6] are discussed. In this paper, we discuss five different types of recursive relationships by further defining the concept of an asymmetric association into hierarchical, circular, and mirrored relationships. We also introduce the concept of a hybrid relationship by combining the concepts of a circular and hierarchical association. By expanding the work of [6], we present a complete taxonomy of recursive relationships and decision rules for checking the validity of those various recursive relationships.

Figure 2 shows a simple entity-relationship diagram containing several recursive relationship types of different semantics. The correct interpretation and validity of these recursive relationships are only possible with both minimum and maximum cardinality. The work presented in our paper will show generalized decision rules for this diagram and any other ER diagrams, and will show what cardinality constraints are allowed and disallowed for all recursive relationships.

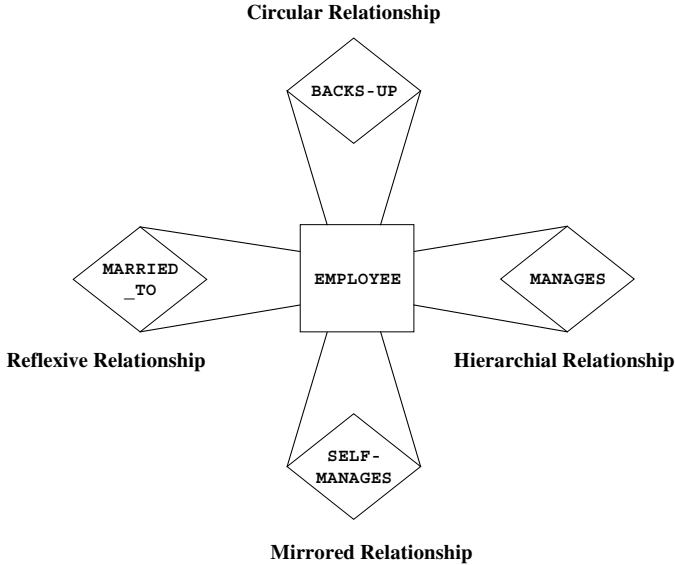


Figure 2: A plausible ER diagram containing various recursive relationships.

Unlike typical other analyses that use only maximum cardinality constraints, we have used both maximum and minimum cardinality constraints in defining the properties and validity criteria. Our analysis yields a complete and comprehensive set of decision rules to determine the structural validity of any recursive relationships. The decision rules use the notions of role uniqueness, path connectivity, and cardinality constraints. These decision rules can be readily applied to real world data models regardless of their complexity. The rules can easily be incorporated into the database modeling and designing process, or extended into case tool implementations.

This paper is organized into five sections. In Section 2 the concepts of semantic and structural validity are discussed and defined. Section 3 presents the taxonomy of recursive relationships and discusses the details of decision rules for checking the structural validity of recursive relationships. Section 4 summarizes all the decision rules and corollaries with examples. Section 5 concludes our paper.

2. Validity

An entity-relationship model is composed of entities, the relationships between entities, and constraints on those relationships. Entities may be chained together in a series of alternating entities and relationships, or may participate singularly with one or more relationships. The connectivity of entities and relationships is called a *path*. Paths are the building blocks of our study in structural validity analysis, and visually define the semantic and structural association that each entity has simultaneously with all other entities or with itself within the path. The terms, structural and semantic validity, are defined as follows:

Definitions: *An Entity Relationship Diagram is structurally valid only when simultaneous consideration of all structural constraints imposed on the model does not imply a logical inconsistency in any of the possible states. An Entity Relationship Diagram is semantically valid only when each and every relationship exactly represents the modeler's concept of the application domain.*

In data modeling, validity can be classified into two types: semantic and structural validity. A semantically valid ERD shows the correct representation of the application domain being modeled. The diagram must communicate exactly the intended concept of the environment as viewed by the modeler. Since the semantic validity is application-dependent, we cannot define generalized validity criteria. Therefore, we do not consider semantic validity in this paper.

The structural validity of an ER diagram is concerned whether or not a given ER diagram contains any constructs that are contradictory each other. An ER diagram with at least one inconsistent cardinality constraint is *structurally invalid*. A structurally valid ERD represents the application semantics in terms of maximum and minimum cardinality constraints. The driving force behind cardinality constraint placement is the semantics of the model. The values and placement of these constraints must be robust enough to convey the business rules exactly intended by the modeler while being consistent with respect to the whole model in order to reflect the real world environment. The model is not just a set of individually constrained relationships pieced together between sets of entities, but a holistic view of the representation domain. Each set of cardinality constraints on a single relationship must be consistent with all the remaining constraints in the model and over all possible states.

In a recursive relationship the association between role groups within a single entity is semantically invalid when the diagram does not reflect the business rules as defined by the user community. A recursive relationship is structurally invalid when the cardinality and participation constraints do not support the existence of data instances as required by the user. A structurally invalid diagram reflects semantically inconsistent business rules.

In order for a model to be valid all the paths in the model must also be valid. Our analysis will investigate those types of structural paths in an entity-relationship diagram that are critical to validity of the entire diagram. Our study will focus primarily on the structural validity of the recursive relationships represented in an ER diagram.

3. Recursive Relationships

Section 3.1 presents our taxonomy of recursive relationships and Section 3.2 presents the complete decision rules for checking the validity of recursive relationships.

3.1. The Taxonomy of Recursive Relationships

A *recursive* (or unary) *relationship* is defined as an association between instances as they take on *roles* within the same entity. Roles play an important part in the examination of structural validity, especially for recursive relationships. A *role* is the action or function that the instances of an entity play in a relationship [12]. In a recursive relationship, a set of instances may take on a single role or multiple roles within the same relationship. Examining these roles allows us to classify all recursive relationships into symmetric or asymmetric associations while further classifying asymmetric relationship types into hierarchical, circular, and mirrored associations. The complete classification of recursive relationships we consider in this paper is shown in Figure 3 as follows:

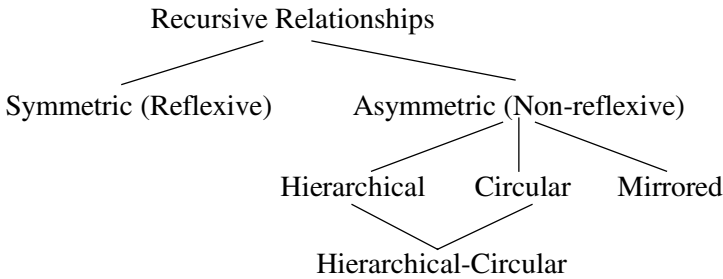


Figure 3: Taxonomy of Recursive relationships

A recursive relationship is *symmetric* or *reflexive* when all the instances participating in the relationship take on a single role and the semantic meaning of the relationship is exactly the same for all the instances participating in the relationship independent of the direction in which it is viewed. These relationship types are called bi-directional. For example, if instance I_1 is associated with another instance I_2 in the same entity through relationship R and the I_2 is associated with I_1 using exactly the same semantics of relationship R then the relationship is symmetric. Relationships such as “dance partner of”, “spouse of”, and “sibling of” are symmetric relationships.

A recursive relationship is *asymmetric* or *non-reflexive* when there is an association between two **different** role groups within the same entity and the semantic meaning of the relationship is different depending on the direction in which the associations between the role groups are viewed. These relationship types are called uni-directional. For example, the instances of an entity **EMPLOYEE** associated though a relationship called **SUPERVISES** contain two roles groups. The role groups are “supervisor employees” and “supervised employees”. In one direction the relationship is viewed as ‘supervisor employees *supervise* supervised employees’ while in the other direction the relationship is viewed as ‘supervised employees *are supervised by* supervisor employees’.

Previous studies [6, 15] of what here is described as a symmetric or reflexive relationship implicitly suggested that the only other relationship type to be investigated would be an asymmetrical relationship. In this study we further classify the concept of an asymmetric relationship type into three *non-reflexive* recursive relationship types: hierarchical, circular, and mirrored relationships.

A recursive relationship is *hierarchical* when a group of instances within the same entity are ranked in grades, orders, or classes, one above another. It implies a beginning (or top) and an end (or bottom) to the ranking scheme of instances. An example of a hierarchical recursive relationship is the entity **EMPLOYEE** and the relationship **SUPERVISES**. In this relationship some employees supervise other employees. There is usually an employee at the top who is not supervised by any other employee and employees at the bottom who do not supervise any employees.

A recursive relationship is *circular* when an asymmetrical recursive relationship has at least one instance that does not comply with the ranking hierarchy. The relationship is unidirectional in that it can be viewed from two directions with different semantic meaning. An example of this occurrence is an entity representing lifeguards positioned around a lake where each lifeguard backs-up the lifeguard to their right when an emergency occurs. For example, lifeguard L *backs-up* lifeguard R while lifeguard R *is-backed-up by* lifeguard L. The relationship in this example is completely circular because no hierarchy was established between lifeguards. We can also develop a hybrid by introducing a hierarchy to the lifeguard instances. If the business rules were changed to allow only senior lifeguards to backup many other lifeguards, then we would have a *hierarchical-circular* relationship. This type of relationship is common among decision-makers where a management employee of a lower rank assumes the responsibilities of a more senior manager during their absence.

Another question that arises in the modeling of recursive relationships is whether an instance can be associated with itself. This event is impossible in relationships above degree one but could happen in special cases of a recursive relationship and we call this special event a *mirrored relationship*. A mirrored relationship exists when the semantics of a relationship allow an instance of an entity to associate with itself through the relationship. For example certain individual contributors in an organization could be self-managed while other individuals report up the management chain. Two relationships would be required to model the supervisory concept. Figure 4 shows the diagram that addresses this issue.

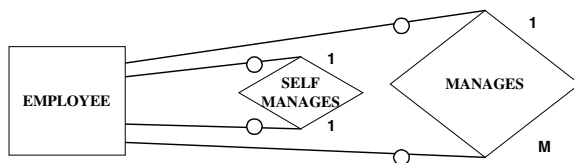


Figure 4: An entity with two asymmetrical recursive relationships addressing self-management issues.

Table 1 summarizes each recursive relationship by its directional properties, the combination of minimum and maximum cardinality constraints, and examples. In our diagrams throughout this paper, we use ‘One (1) and Many (M)’ notation for maximum cardinality and the symbols “#” and “()” to indicate mandatory and optional cardinality [13], respectively. Also, the notation |E| represents the number of instances in entity E.

Type of Relationship		Direction of Relationship	Participation Constraints	Cardinality Constraints	Example Relationship	Example Role(s)
Symmetrical (Reflexive)		Bi-Directional		$1 \text{ --- } 1$ $M \text{ --- } N$	Spouse of	Person
Asymmetrical (Non-Reflexive)	Hierarchical	Uni-Directional		$1 \text{ --- } M$ $1 \text{ --- } 1$	Supervises Is Supervised By	Manager Employee
				$M \text{ --- } N$	Supervises Is Supervised By	Manager Employee
	Circular	Uni-Directional		$1 \text{ --- } 1$	Backs Up Is Backed Up By	Lifeguard
	Hierarchical Circular	Uni-Directional		$1 \text{ --- } 1$ $1 \text{ --- } M$	Backs Up Is Backed Up By	Decision-Maker
				$M \text{ --- } N$	Supervises Is Supervised By	Manager Employee
	Mirrored	Uni-Directional		$1 \text{ --- } 1$	Self Manages Manages Self	CEO

Table 1: Valid Recursive Relationship Types as they relate to the cardinality constraints

3.2 Decision Rules for Validity Checking

In this section, we analyze recursive relationships from the point of view of cardinality and participation constraints. Table 1 pairs each valid set of constraints with the different recursive relationship types. For example, symmetrical relationships are supported by ‘One-to-One’ or ‘Many-to-Many’ cardinality constraints coupled with either ‘Optional-Optional’ or ‘Mandatory-Mandatory’ participation constraints. Only these constraint combinations are valid for a symmetrical relationship. These constraints along with others also are valid in hierarchical and hierarchical-circular relationships, as shown in the table. We review all combinations of constraints and identify the valid recursive relationship types discussed in Section 3.1, and shown in Table 1, through examples and discussion. Those minimum and maximum cardinality constraints that yield invalid structures are demonstrated though formal proofs.

3.2.1 One-to-One Recursive Relationships

There are three cases that require investigation in one-to-one recursive relationships. They are ‘mandatory-mandatory’, ‘optional-optional’, and ‘optional-mandatory’. The ‘mandatory-optional’ case is a reverse image of the ‘optional-mandatory’ case and does not require further analysis.

1:1 Mandatory-Mandatory

This type of relationship is valid for a symmetrical relationship. In a ‘mandatory-mandatory’ relationship each instance of the role group must participate fully in the relationship. When the relationship is 1:1, each instance must be paired with one and only one other instance. An example of this symmetrical pairing of instances is shown in Figure 5 representing of a group of married persons with the relationship `MARRIED_TO`. Every person in the group is married to one and only one other person in the group.

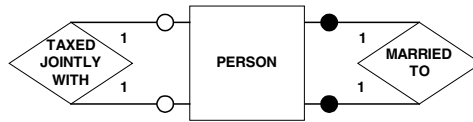


Figure 5: Valid Recursive Relationships.

1:1 Optional-Optional

In an ‘optional-optional’ relationship each instance of the role groups can optionally participate in the relationship. This case makes no restrictions on the number of roles that is contained in the entity, so both symmetrical and asymmetrical relationships are supported. In a ‘one-to-one’ symmetrical relationship, instances of the role group are paired with one and only one other instance in the same role group. If the mandatory constraint is removed from the relationship then additional non-participating instances can be included in the entity without effecting the validity. In Figure 4 married individual have the option of being taxed either jointly or separately. This is an example of a symmetric relationship where the entity is `PERSON` and the relationship is `TAXED_JOINTLY_WITH` can take on a minimum cardinality constraint of ‘mandatory-mandatory’ or ‘optional-optional’ when the maximum cardinality is ‘one-to-one’. For the ‘mandatory-mandatory’ case all of the instances in `PERSON` must file their taxes jointly with their spouse while in the ‘optional-optional’ case, shown in Figure 5 at least one pair of married persons decided to file their taxes separately.

1:1 Mandatory-Optional or Optional-Mandatory

In a ‘mandatory-optional’ relationship each instance of one role group must participate in the relationship while the instances of the other role group can optionally participate in the relationship. This implies the existence of two different role groups in the entity and therefore the relationship must be asymmetric. Symmetric relationships of any type cannot support ‘mandatory-optional’ relationships because an instance participating in a symmetric relationship is mapped to another instance in the same role group. By the definition of a symmetrical relationship the instance being ‘mapped to’ must also have the reflexive property of being able to map back to its paired instance. This would be impossible if the participation is optional. Asymmetrical relationships also have a similar instance-mapping problem, as we will demonstrate.

Theorem 1: A 1:1 recursive relationship structure with a minimum cardinality constraint of ‘mandatory-optional’ is an *invalid* structure.

Proof: Role group 1 ($rg1$) contains I_j instances (where $j = 1, 2, 3, \dots, n$) and represents all the instances in entity E, therefore,

$$|rg1| = |E| = n \quad (1)$$

Because of the mandatory participation, each instance in $rg1$ is mapped totally to role group 2 ($rg2$) and $|rg2|$ is at least equal to n , therefore,

$$|rg2| \geq n \quad (2)$$

If the instances of $rg2$ are only partially mapped to the instances of $rg1$ then the instances in $rg2$ may contain at least one additional instance that is not mapped to instances of $rg1$. Therefore, in the strictest form

$$|rg2| > n \quad (3)$$

$Rg2$ also represents all the instances in entity E, therefore from (3) we can derive

$$|E| > n \quad (4)$$

Equations (1) and (4) are inconsistent because $|E|$ can not be equal to n and greater than n , therefore the structure is invalid. ♦

The proof of Theorem 1 demonstrates that a 1:1 recursive relationship with a minimum cardinality of constraint of ‘mandatory-optional’ or ‘optional-mandatory’ is invalid for both symmetrical and asymmetrical relationships.

Rule 1: Only 1:1 recursive relationships with mandatory-mandatory or optional-optional cardinality constraints are structurally valid.

Corollary 1: All 1:1 recursive relationships with mandatory-optional or optional-mandatory cardinality constraints are structurally *invalid*.

3.2.2 One-to-Many Recursive Relationships

There are four cases requiring investigation in one-to-many recursive relationships. The minimum cardinality constraints are mandatory-mandatory, mandatory-optional, optional-mandatory, and optional-optional. ‘One-to-many’ recursive relationships are always asymmetrical relationships because the non-reflexive relationship requires two role groups.

One-to-Many Mandatory-Mandatory

In a ‘one-to-many’ recursive relationship the ‘mandatory-mandatory’ case is invalid as stated in the following rule and demonstrated in the following proof.

Theorem 2: A ‘One-to-Many’ recursive relationship where the both minimum cardinality constraints are mandatory is an invalid structure.

Proof: Given that role group 1 and 2 are both contained in entity E and participate fully in the relationship R. Because of the 1:M constraint placed on R, the relationship R must support the case of *rg1* containing fewer instances than *rg2*, therefore, in the strictest form

$$|rg1| < |rg2| \quad (1)$$

Because of the mandatory constraint on R the instances of *rg1* represent all the instances in E, therefore,

$$|rg1| = |E| \quad (2)$$

Also because of the mandatory constraint on R the instances of *rg2* represent all the instances in E, therefore,

$$|rg2| = |E| \quad (3)$$

$$\text{If } |rg1| = |E| \text{ and } |rg2| = |E|$$

$$\text{Then } |rg1| = |rg2| \quad (4)$$

Equations (1) and (4) are inconsistent therefore the structural representation is invalid. ♦

One-to-Many Mandatory-Optional

Also in the ‘one-to-many’ recursive relationship, the ‘mandatory-optional’ case is invalid as stated in the following rule and demonstrated in the following proof.

Theorem 3: Any ‘One-to-Many’ recursive relationship where the minimum cardinality constraint is mandatory on the ‘One’ side and optional on the ‘Many’ side is an invalid structure.

Proof: Consider role groups 1 and 2 that are contained in entity E and that participate in a 1:M relationship R. If *rg1* on the ‘One’ side of a ‘One-to-Many’ relationship is mapped with optional participation to *rg2* on the ‘Many’ side, then the relationship R must support the case of *rg1* containing fewer instances than *rg2*. Therefore, in the strictest form

$$|rg1| < |rg2| \quad (1)$$

Because of the mandatory cardinality constraint on the role group 1 the number of instances in *rg1* must equal the number of instances in entity E. Therefore,

$$|rg1| = |E| \quad (2)$$

Combining equations (1) and (2) implies that $|E| < |rg2|$ but this is inconsistent because the number of instances in a role group can not exceed the total number of instances in the entity containing that role group. ♦

One-to-Many Optional-Mandatory

A hierarchical-circular recursive relationship type supports the occurrence of the ‘optional-mandatory’ case in a ‘One-to-Many’ recursive relationship. An example would be where there are two roles being played by the instances of the entity. First, every

instance participates as an employee and employees are expected to be able to make decisions about the area of their expertise. Some employees have more than one area of expertise and are required to backup other employees during their absence. The relationship is hierarchical because all employees participate in the relationships of being backed-up at the bottom of hierarchy but only some employees at the top of the hierarchy backup all other employees. The relationship is circular because some one in the tree structure circles back to backup the instance at the top. Figure 6 is an example of this type of relationship. This relationship is similar to the lifeguard example but allowing an instance to backup more than one other instance.

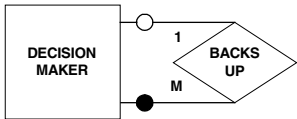


Figure 6: An example of a One-to-Many Optional-Mandatory Recursive Relationship

One-to-Many Optional-Optional

Howe [10, p. 137] implies that “the most senior employee is regarded as self-supervised” and allows the ‘many’ side to be mandatory. Using this method only implies that an employee can be self-managed and may lead to a different interpretation. We feel the concept of role uniqueness should be explicitly stated in the diagram and the more appropriate way to model the example would be with two relationships, both being ‘optional-optional’. One relationship specifically addressing the self-management issue. Figure 6 shows a suggested diagram.

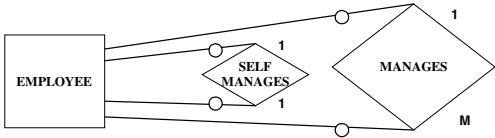


Figure 7: An entity with two asymmetrical recursive relationships addressing the self-management issue.

Figure 7 demonstrates the validity of the ‘optional-optional’ case of a 1:M relationship. It is optional for the manager role group because only some instances are managers while others are not. It is optional for the employee role group because as previously stated at least one manager at the higher level is an employee that is not managed by other instances in the entity. From Theorems 2 and 3 proofs and the accompanying discussion the following rule and corollary can be stated about 1:M relationships.

Rule 2: *For 1:M or M:1 recursive relationships optional-optional cardinality is structurally valid.*

Rule 3: *For 1:M recursive relationships of the hierarchical-circular type, optional-mandatory cardinality are structurally valid.*

Corollary 2: *All 1:M recursive relationships with mandatory-mandatory cardinality constraints are structurally **invalid**.*

Corollary 3: *All 1:M recursive relationships with mandatory-optional cardinality constraints are structurally **invalid**.*

3.2.3 Many-to-Many Recursive Relationships

There are three cases requiring investigation in many-to-many recursive relationships. The minimum cardinality constraints are mandatory-mandatory, mandatory-optional, and optional-optional.

Many-to-Many Mandatory-Mandatory

A ‘mandatory-mandatory’ many-to-many relationship is valid for a symmetrical relationship while as stated before the asymmetrical recursive relationship can not be totally mandatory. In a ‘mandatory-mandatory’ relationship each instance of a role group must participate fully in the relationship. When the relationship is a ‘many-to-many’ relationship then each instance must be paired with one or more instances in the role group. Because they are symmetric, each pairing is reflexive meeting the requirement of a ‘mandatory-mandatory’ relationship. A valid example would be an entity of PERSON composed of groups of brothers and sisters with the SIBLING_OF as the relationship.

Many-to-Many Mandatory-Optional

In the example of a hierarchical tree for a ‘many-to-one’ relationship using the entity EMPLOYEE with different levels of management, we showed that the minimum cardinality constraint is ‘optional’ on both sides of the relationship. This occurred because some employees were at the bottom of the management chain and did not manage anyone, and one employee, the highest manager, was at the top of the chain not managed by any one. If we change this example to a company that is completely employee-owned and some of these employees form the board of directors that manage the senior manager, then this is an example of a ‘Many-to-Many’, Mandatory-Optional recursive relationship. In this modify example some employees (but not all) take on the role of manager satisfying the optional side of the relationship. The mandatory side of the relationship is satisfied because all employees are managed even the most senior manager and the owner-employees. This is an example of a ‘Many-to-Many’ relationship because the senior manager is managed by more than one owner-employee and manages more than one employee in the hierarchy. This type of relationship with these constraints is valid.

Many-to-Many Optional-Optional

Our earlier example of an entity PERSON composed of groups of only brothers and sisters with the SIBLING_OF as the relationship is a valid ‘Mandatory-Mandatory’ ‘Many-to-Many’ recursive relationship and can serve a model for the totally optional case. If we add individuals to the entity PERSON that have no brothers or sisters then these individuals can not participate in the relationship SIBLING_OF. Of more importance is that they do not participate on both sides of the relationship because they have no brothers or sisters, and no other instance has them as brothers or sister. The relationship is optional on both sides. This is an example of a valid symmetric relationship.

Rule 4: *All recursive relationships with many-to-many maximum cardinality are structurally valid regardless of minimum cardinality constraints.*

Rule 5: *All recursive relationships with optional-optional cardinality are structurally valid.*

4. Summary of Decision Rules with Examples

From Section 3.2 we have developed five decision rules for determining the validity of a recursive relationship and three corollaries that quickly identifies invalid recursive relationships. Table 2 summarizes each validity rule and corollary for recursive relationships with an example. The relationship types that are addressed by each rule are also included.

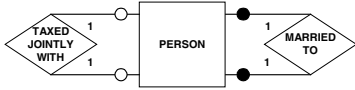
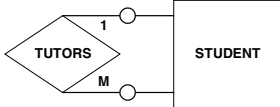
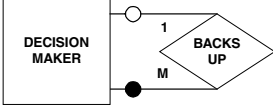
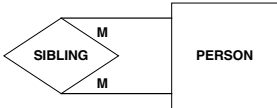
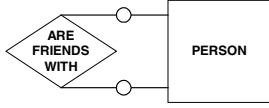
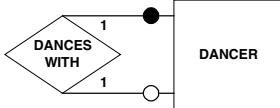
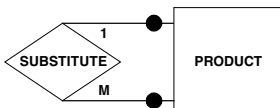
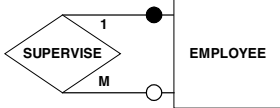
Validity Rules for Recursive Relationships	Valid Example
1. Only 1:1 recursive relationships with mandatory-mandatory or optional-optional cardinality constraints are structurally valid. Valid for symmetrical and completely circular relationships.	
2. For 1:M or M:1 recursive relationships optional-optional cardinality are structurally valid. Valid for asymmetrical relationships only.	
3. For 1:M recursive relationships of the hierarchical-circular type, optional-mandatory cardinality are structurally valid. Valid for hierarchical-circular relationships only.	
4. All recursive relationships with many-to-many maximum cardinality are structurally valid regardless of minimum cardinality constraints. Valid for symmetrical, hierarchical, and hierarchical-circular relationships.	
5. All recursive relationships with optional-optional cardinality are structurally valid. Valid for symmetrical and asymmetrical relationships.	
Validity Corollaries for Recursive Relationships	Invalid Example
1. All 1:1 recursive relationships with mandatory-optional or optional-mandatory cardinality constraints are structurally invalid .	
2. All 1:M or M:1 recursive relationships with mandatory-mandatory, cardinality constraints are structurally invalid .	
3. All 1:M or M:1 recursive relationships with mandatory participation constraint on the 'One' side and an optional participation constraint on the 'Many' constraints are structurally invalid .	

Table 2: Summary of Validity Rules for Recursive Relationships with Examples

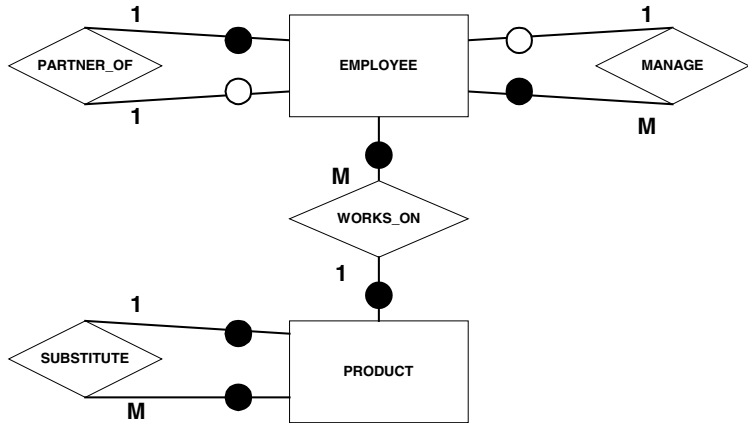


Figure 1: An ERD with Invalid Recursive Relationships
(Reproduced here for the convenience of the reader)

In Figure 1 of this paper the entity-relationship diagram contained three recursive relationships: **PARTNER_OF**, **MANAGE**, and **SUBSTITUTE**. **PARTNER_OF** is invalid because it violates Corollary 1. It was shown in this paper by formal proof presented for Theorem 1 that a ‘One-to One’ recursive relationship with a ‘Mandatory-Optional’ participation constraint is invalid. **SUBSTITUTE** is invalid because it violates Corollary 2. The proof of Theorem 2 states that a ‘One-to-Many’ recursive relationship with a ‘Mandatory-Mandatory’ participation constraint is invalid. **MANAGE** is invalid because it violates Corollary 3. The proof of Theorem 3 states that a ‘One-to-Many’ recursive relationship with a ‘Mandatory-Optional’ participation constraint is invalid. ERDs containing these recursive relationships are invalid.

5. Conclusion

In this paper, we have presented the complete classification of recursive relationships and the criteria that contribute to the validity of modeling recursive relationships within the entity-relationship (ER) diagram. We have first classified all recursive relationships into symmetric and asymmetric. Asymmetric recursive relationships are further classified into hierarchical, circular, and mirrored. We have identified their directional properties and their validity in terms of maximum and minimum cardinality constraints. Unlike typical other analyses that use only maximum cardinality constraints, we have used both maximum and minimum cardinality constraints in defining the properties and validity criteria. Our analysis has yielded a complete and comprehensive set of decision rules to determine the structural validity of any ER diagram containing recursive relationships. Our decision rules used notions of role uniqueness, path connectivity, and cardinality constraints. Five rules and three corollaries were

established to determine structural validity of recursive relationships. Section 4 summarized each rule and corollary with valid and invalid examples.

The contribution of this paper is the presentation of a complete taxonomy of recursive relationships with their properties as well as the decision rules for their validity. We believe that these decision rules can be readily applied to real world data models regardless of their complexity. The rules can easily be incorporated into the database modeling and designing process, or extended into case tool implementations.

References

- [1] Barker, R. *CASE*METHOD®: Entity Relationship Modeling*. Addison-Wesley, New York 1990.
- [2] Boehm, Barry W., Software Engineering, *IEEE Transactions on Computers*, 25(12)1226-1241, December 1976.
- [3] Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, Inc. Englewood Cliffs, NJ, 1981.
- [4] Chen, Peter, "The Entity-Relationship Model -- Toward a Unified View of Data", *ACM Transactions on Database Systems*, 1(1)9-36, March 1976.
- [5] Daly, E. B., Management of Software Engineering, *IEEE Transactions on Software Engineering*, 3(3)229-242, May, 1977
- [6] Dullea, James and Il-Yeol Song, "An Analysis of Structural Validity in Recursive and Binary Relationships in Entity Relationship Modeling", *The Proceedings of the Fourth International Conference on Computer Science and Informatics*, Volume 3, pp 329-334, Research Triangle Park, NC, October 23-28, 1998.
- [7] Dullea, James and Il-Yeol Song, "An Analysis of Structural Validity of Ternary Relationships in Entity-Relationship Modeling," *Proceedings of Seventh International Conference on Information and Knowledge Management (CIKM '98)*, Nov. 3-7, 1998, Washington, D.C., pp. 331-339.
- [8] Elmasri, Ramez and Shamkant B. Navathe. *Fundamentals of Database Systems*, 2nd Ed., The Benjamin/Cummings Publishing Co, Inc., Redwood City, CA, 1994.
- [9] Fagan, M. *Design and Code Inspections and Process Control in the Development of Programs*, IBM Report IBM-SDD-TR-21-572, December 1974.
- [10] Howe, D. R., *Data Analysis for Data Base Design*, 2nd Ed., Edward Arnold, London, GB, 1989.
- [11] Trevor H. Jones and Il-Yeol Song, "Analysis of Binary/ternary Cardinality Combinations in Entity-Relationship Modeling," *Data & Knowledge Engineering*, Vol 19, No. 1(1996), pp. 39-64.
- [12] Korth, Henry F., and Abraham Silberschatz, 1997. *Database System Concepts*, 3rd Ed., McGraw-Hill, Inc., New York.
- [13] Song, Il-Yeol, Mary Evans, and E. K. Park, "A Comparative Analysis of Entity-Relationship Diagrams", *Journal of Computer & Software Engineering*, 3(4)427-459, 1995.
- [14] Teorey, Toby J., *Database Modeling and Design - The Entity-Relationship Approach*, 3rd ed., Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1998.
- [15] Tillman, George, *A Practical Guide to Logical Data Modeling*, McGraw-Hill, Inc., New York, 1993.

Extending Functional Dependencies in Indefinite Sequence Relations

Wilfred Ng

Department of Computing
Hong Kong Polytechnic University
csshng@comp.polyu.edu.hk

Abstract. We introduce the concept of an indefinite sequence relation r over a relation schema R in order to capture the semantics of sequence data having indefinite values. Indefinite information may arise in a relation containing data from a mobile environment or from a multi-database system. An indefinite tuple allows an attribute value to contain a set of values, which can be employed to represent disjunction of information. A definite sequence relation extracted from r , while maintaining its ordering, has all indefinite cells replaced with just one of the indefinite values. We establish a formal hierarchical structure over the class of indefinite sequence relations having a fixed schema and cardinality, which can be employed to classify the relations into different levels of precision. A functional dependency (FD) f satisfies in r if there is a definite sequence relation over R , extracted from r , that satisfies f as the same way a conventional relation does. Our main result shows that FDs in this context have the following two important features. First, Lien and Atzeni's axiom system [23,3] for FDs in incomplete relations, rather than the well-known Armstrong's axiom system, is sound and complete for FDs in indefinite sequence relations. It implies that the established results for FDs in incomplete relations, such as the closure and implication algorithms, can be extended to FDs in indefinite sequence relations. Second, the satisfaction for FDs is non-additive, meaning that in a given relation the satisfaction of each f in F may not lead to the the satisfaction of F .

1 Introduction

It is generally accepted that sequence data is useful in many database applications [27,36,34] due to the fact that many real-life information contains logical ordering relationships between data items. For example, the recent investigation of sequence database systems [34], temporal database systems [24] and those applications involving linear ordered data [25,22,31,32]. Indefinite information may arise in a relation containing data from a mobile environment or from a multi-database system [4]. We introduce the notion of an indefinite sequence relation in order to cater for the situation when the information arriving in some logical ordering from different sources and therefore may be indefinite. Components of tuples in an indefinite sequence relation is a tuple whose values may be a sets of

values rather than a single value. An indefinite tuple t can also be regarded as a set of fuzzy tuples that are extracted from t and have *identical memberships* [35].

Example 1. As a motivating example we let $CURRENCY\_TABLE = \{TIME, EXCHANGE\_RATE\}$ be a relation schema, where the semantics is that in a rapid changing currency market at a certain TIME the EXCHANGE RATE from U.S. dollars to H.K. dollars is obtained and recorded. Using the information from this relation a currency broker is able to make a quick investment decision. The data of the exchange rate in this case may be obtained from different sources in the world and sometimes the quantity of it is quite large in a day. In order to make the received data more easy referencing, we do not distinguish those values arriving at similar time. On the other hand, we may still want to keep them as a sequence of records to indicate the approximate time of arrival. So we collapse the data into a set of (equally likely) values and thereby form an indefinite tuple. Collapsing “primitive” data in this way can also be used in data cleaning or compacting processes to control the granularity of data, for instance, prior to its placement within a data warehouse [18]. A sequence of indefinite tuples t_1, t_2 and t_3 is recorded as shown in Figure 1.

	<i>TIME</i>	<i>EXCHANGE_RATE</i>
t_1	{0900, 0902, 0905}	{7.77, 7.78}
t_2	{1000, 1011}	{7.55}
t_3	{1200, 1211, 1213}	{7.60, 7.67}

Fig. 1. An example of an indefinite sequence relation of financial data

We note that the tuple ordering is significant to database design and implementation in both the conventional or advanced DBMSs. At the logical level, in Example 1 we have shown that tuple ordering can be employed to represent the temporal semantics of the data. This can be further extended in a more general way to model other kinds of ordered data such as spatial data [30]. At the physical level, the notion of tuple ordering allows users to derive more accessing and processing methods based on this linear structuring. In this respect it is interesting to see the discussion in [1] for the formalisation of tuple sequences in conventional relational DBMS, and the recent investigation in [28] for using tuple ordering to enhance the performance of object-relational DBMSs in a parallel environment.

Naturally, the notion of “definiteness” is central to the semantics of an indefinite sequence relation. Our approach to formalise this notion in this context is straightforward and intuitive: given a schema, the smaller the cardinality of all the cell values of a tuple, the “more definite” (or equivalently, “less indefinite”) that the tuple is. We extend this notion to relations and establish a formal structure called *precision hierarchy* over the set of possible indefinite sequence relations. The hierarchy allows us to classify relations into different levels according to their degree of precision. The higher the level of an indefinite relation, the more definite that the relation is.

Functional dependencies (FDs) are commonly recognized as the most fundamental integrity constraints arising in practice in conventional relational databases [38,3]. We formalize the notion of FDs being satisfied in indefinite sequence relations. The semantics of an indefinite sequence relation, r over a relation schema R , is defined using the *possible world approach*. The set of *possible worlds* relative to r , denoted by $poss(r)$, is the set of all definite sequence relations emanating from a set of possible elements in an indefinite cell. An FD f is said to be satisfied in r , if there exists a possible world in $poss(r)$ that satisfies f , when using the standard definition of the satisfaction of an FD holding in the definite sequence relation. For instance, referring to the relation in Figure 1, we can check that it satisfies the FD $TIME \rightarrow EXCHANGE\_RATE$. The enforcement of this FD in the relation brings the following two benefits. First, it ensures that any two tuples arriving at an exact time do not have their corresponding sets of exchange rates being disjoint in the relation, asserting the existence of a consistent possible world. Second, we can employ data mining techniques adapted to indefinite sequence relations so as to discover the existence of a consistent possible world that satisfies (approximately) some ordered trends (c.f. Ordered FDs discussion in [31,32]).

In the database literature, the *implication problem* is an important issue arising from investigating data dependencies [16,12]. So we herein investigate the implication problem of FDs for indefinite sequence relations. Our approach is to establish a set of *inference rules* F which constitutes the *axiom system* $\mathcal{A}$, which is a common tactics to tackle the implication problem of data dependencies [38,3]. We can use the rules of $\mathcal{A}$ to *derive* a functional dependency f from F . We call $\mathcal{A}$ *sound and complete*, if we can prove that, by using $\mathcal{A}$, f can be derived from F , if and only if a relation r satisfies F , it also satisfies f .

This research also relates to the fuzzy relational model [33,35], which used fuzzy set theory to deal with uncertain or imprecise information. We note that another interesting approach is to use the rough set theory to deal with the indefiniteness [20]. We also note that the probability theory can also be used as a foundation for indefinite information by imposing some pre-defined distribution of data [7]. However, we emphasise that the semantics of fuzzy FDs in fuzzy relations [5,33] is different from that of FDs in indefinite sequence relations, even we do not take into the fact that tuple ordering is an essential part in an indefinite sequence relation. Our definition of FDs using the possible world approach is, in general, less restrictive than the definition of fuzzy FDs in fuzzy relations, for example the one being based upon the equivalence classes model [35]. For the sake of easy comparison, we now view an indefinite sequence relation having no repeated tuples as a special case of fuzzy relation r [21], then it can be shown that if $r \models f'$, where f' is a fuzzy functional dependency, then $r \models f$, where f is a corresponding functional dependency defined in an indefinite sequence relation. However, the converse is not true. We illustrate this difference by the following example.

Example 2. For the sake of simplicity we do not distinguish a sequence and a set of two tuples t_1 and t_2 in r given in Figure 2 (a). We also assume all the values in a set are equally likely, which can be regarded as a special case of a fuzzy set whose elements having identical membership values. So the semantics of the fuzzy FD $A \rightsquigarrow B$ in this case can be interpreted as follows, whenever $t_1[A] \sim t_2[A]$ then $t_1[B] \sim t_2[B]$, where $X \sim Y$ is defined as $X \cap Y \neq \emptyset$. It can be checked that r satisfies $A \rightarrow B$, since there exists a possible definite sequence relation r_{def} (i.e. a consistent possible world) in Figure 2 (b), extracted from r , satisfying $A \rightarrow B$. However, r does not satisfy $A \rightsquigarrow B$, since $t_1[A] \sim t_2[A]$ but $t_1[B] \not\sim t_2[B]$. Informally, it means that to certain extent $t_1[A]$ is *equal* to $t_2[A]$ (i.e. $\{0\} \sim \{0, 1\}$) but $t_1[B]$ is not *equal* to $t_2[B]$ (i.e. $\{2\} \not\sim \{3\}$).

$r =$	<table style="border-collapse: collapse; margin: auto;"> <tr> <th style="border: none;"></th> <th style="border: none;">A</th> <th style="border: none;">B</th> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">t_1</td> <td style="border: 1px solid black; padding: 2px;">$\{0\}$</td> <td style="border: 1px solid black; padding: 2px;">$\{2\}$</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">t_2</td> <td style="border: 1px solid black; padding: 2px;">$\{0, 1\}$</td> <td style="border: 1px solid black; padding: 2px;">$\{3\}$</td> </tr> </table>		A	B	t_1	$\{0\}$	$\{2\}$	t_2	$\{0, 1\}$	$\{3\}$	$r_{def} =$	<table style="border-collapse: collapse; margin: auto;"> <tr> <th style="border: none;"></th> <th style="border: none;">A</th> <th style="border: none;">B</th> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">t_1</td> <td style="border: 1px solid black; padding: 2px;">$\{0\}$</td> <td style="border: 1px solid black; padding: 2px;">$\{2\}$</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">t_2</td> <td style="border: 1px solid black; padding: 2px;">$\{1\}$</td> <td style="border: 1px solid black; padding: 2px;">$\{3\}$</td> </tr> </table>		A	B	t_1	$\{0\}$	$\{2\}$	t_2	$\{1\}$	$\{3\}$
	A	B																			
t_1	$\{0\}$	$\{2\}$																			
t_2	$\{0, 1\}$	$\{3\}$																			
	A	B																			
t_1	$\{0\}$	$\{2\}$																			
t_2	$\{1\}$	$\{3\}$																			
	(a)		(b)																		

Fig. 2. An example of a relation satisfying a functional dependency $A \rightarrow B$ but not the corresponding fuzzy functional dependency $A \rightsquigarrow B$

The rest of the paper is organized as follows. In Section 2 we give a formal definition of indefinite sequence relations on which we study a formal structure called *precision hierarchy* via the notion of indefiniteness. In Section 3 we define the semantics of FDs being satisfied in indefinite sequence relations and show that Lien and Atzeni's axiom system is sound and complete for FDs in this context. In Section 4, we present our investigation that the satisfaction for FDs in indefinite sequence relations is not additive, which is different from that in the conventional case. In Section 5 we give our concluding remarks.

Throughout this paper we make use the following notation.

Definition 1. (Notation) Let S and T be sets, then $|S|$ denotes the *cardinality* of S , $S \subseteq T$ denotes *inclusion*, $S \subset T$ denotes *proper inclusion* and $\mathcal{P}(S)$ denotes the *non-empty finite powerset* of S . We denote the k term Cartesian product $S \times S \times \dots \times S$ by S^k and the singleton $\{A\}$ simply by A when no ambiguity arises. Some usual set notations [15] also apply to a sequence, for instance $|T|$ means the length of the sequence T . We take $A \in \langle A_1, \dots, A_n \rangle$ to mean $A \in \{A_1, \dots, A_n\}$ and $\langle A_1, \dots, A_n \rangle \subseteq \langle B_1, \dots, B_m \rangle$ where $n \leq m$, to mean $\{A_1, \dots, A_n\} \subseteq \{B_1, \dots, B_m\}$. We may also write $A_1 \dots A_n$ instead of $\langle A_1, \dots, A_n \rangle$.

2 The Indefinite Sequence Relation Model

We now formalize the notion of an indefinite sequence relation, which consists a sequence of tuples in which allow components of tuples to be non-empty sets of values rather than just a single value as is the case when the information is definite.

Definition 2. (Relation Schema and Indefinite Sequence Relations)

Let $\mathcal{U}$ be a countable set of attributes and $\mathcal{D}$ be a countable set of domain values. A *relation schema* R is a finite set of attributes in $\mathcal{U}$. An (*indefinite*) *tuple* t over R is a total mapping from R into $\mathcal{P}(\mathcal{D})$ such that $\forall A \in R, t(A) \in \mathcal{P}(A)$. A tuple t over R is *definite* if $\forall A \in R, |t(A)| = 1$, i.e. $t(A)$ is a singleton, where $|t(A)|$ denotes the cardinality of $t(A)$. An *indefinite sequence relation* (or simply a relation) over R is a finite (possible empty) sequence of indefinite tuples over R . A relation r over R is *definite* if all of its tuples are definite.

The motivation for defining a relation as a sequence of tuples is that in many database applications, such as those involving temporal or financial information, need the support of sequence data in a DBMS [24,37,29] as shown in Example 1. From now on, we let R be a relation schema, r be a relation over R and $t \in r$ be an indefinite tuple.

Definition 3. (The Set of Possible Definite Sequence Relations) The set of *possible* definite tuples of a tuple t , denoted by $poss(t)$, is the set of tuples given by $\{u \mid u \text{ is definite and } \forall A \in R, u[A] \in t[A]\}$. The set of *possible definite sequence relations* arising from a relation $r = \langle t_1, \dots, t_n \rangle$, denoted by $poss(r)$, is the set of definite sequence relations given by $\{s \mid s = \langle u_1, \dots, u_n \rangle \text{ and } u_1 \in poss(t_1), \dots, u_n \in poss(t_n)\}$.

The projection operation onto a relation is similar to the conventional case except that the cardinality of the projected relation is preserved to be $|r|$ in the result. This also implies that the number of duplicated tuples in the projected relation will not be less than that of the original relation. This assertion is not true in conventional relations.

Definition 4. (Projection) The *projection* of a tuple t onto a set of attributes $X \subseteq R$, denoted by $t[X]$, is the restriction of t to X . The projection of a relation $r = \langle t_1, \dots, t_n \rangle$ over R onto X , denoted by $\pi_X(r)$, is defined by $\pi_X(r) = \langle t_1[X], \dots, t_n[X] \rangle$.

The semantics of a set $t[A]$, where t is a tuple over R and $A \in R$, are that all the values $a \in t[A]$ are possible and thus they constitute an indefinite tuple. Using this idea we can compare the “definiteness” of two tuples. We remark that $t[A]$ cannot be empty as we do not allow $\emptyset$ in defining the powerset operator.

Definition 5. (Less Definite Tuples) Let t_1 and t_2 are two tuples over R . t_1 is said to be *less definite* than t_2 , denoted as $t_1 \sqsubseteq t_2$, if $\forall A \in R, t_2[A] \subseteq t_1[A]$

The above definition provides a formal metric to measure how “definite” a tuple is. Intuitively, the smaller the cardinality of all the cell values of a tuple, the “more definite” (or equivalently, “less indefinite”) that the tuple is. For example, if we have two possible exchange rates for a currency then we have more definite information than if we have three possible exchange rates for that currency. We first give a semantic definition of “definiteness” of a relation and next characterize it syntactically in terms of its tuples.

Definition 6. (Less Definite Relations) Let r_1 and r_2 are two relations over R . r_1 is said to be less definite than r_2 , denoted as $r_1 \sqsubseteq r_2$, if $\text{poss}(r_2) \subseteq \text{poss}(r_1)$.

It is easy to see that $\text{poss}(r)$ consists those relations having equal number of tuples. So $\text{poss}(r_2) \subseteq \text{poss}(r_1)$ is valid only if $|r_1| = |r_2|$. Importantly, if $r_1 \sqsubseteq r_2$, then $|r_1| = |r_2|$. The predicate, $\sqsubseteq$, is a partial order in the set of relations over R if we assume the constraint that the cardinality of a relation is fixed to be some non-negative integers n . We let $\text{rel}(R, n)$ the set of all relations over R whose cardinality is n . It is easy to check that $|\text{rel}(R, n)| = |\mathcal{P}(\mathcal{D})|^{n|R|}$. The following proposition gives a syntactic characterization of the notion of less definite relations.

Proposition 1. Let $r_1, r_2 \in \text{rel}(R, n)$, t_i and t'_i be the i th tuples in r_1 and r_2 , respectively. $r_1 \sqsubseteq r_2$ if and only if $t_i \sqsubseteq t'_i \forall i \in \{1, \dots, n\}$. $\square$

In order to classify the degree of precision of relations in a formal manner, we now define a *precision hierarchy* on $\text{rel}(R, n)$ with respect to $\sqsubseteq$, whose top level (T_1) consists of all the definite sequence (i.e. the most precise) relations choosing from $\text{rel}(R, n)$ and the bottom level (T_b) consists of an (unique) indefinite sequence (i.e. the least precise) relation consisting n (identical) tuples $\langle \mathcal{D}, \dots, \mathcal{D} \rangle$. We denote by $\text{part}(R, n)$ a *partition* of $\text{rel}(R, n)$, which is a set of pairwise disjoint non-empty subsets of $\text{rel}(R, n)$ such that $\bigcup_{T \in \text{part}(R, n)} T = \text{rel}(R, n)$, and call an element $T \in \text{part}(R, n)$ a *precision level* of r . A *precision hierarchy* of $\text{rel}(R, n)$ is a linearly ordered partition induced by $\sqsubseteq$. In Figure 3 we show a simple example of a precision hierarchy with $R = \{A\}$ and $n = 2$.

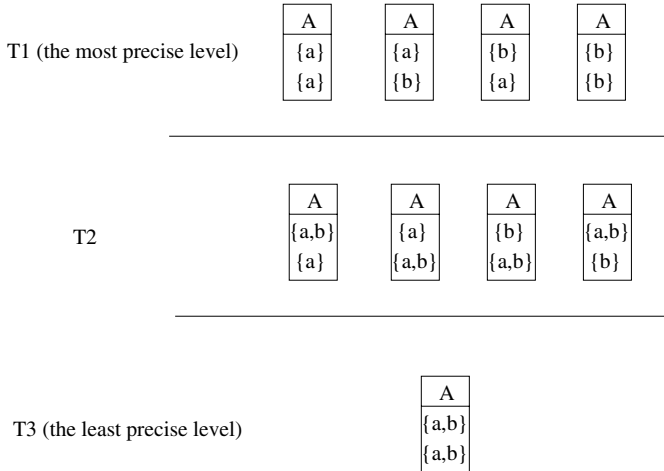


Fig. 3. An Example of the precision hierarchy on $\text{rel}(A, 2)$

Definition 7. (Precision Hierarchy of Relations) A *precision hierarchy* of $rel(R, n)$ is a linearly ordered set $\langle part(R, n), \leq \rangle$ generated by the following algorithm.

Algorithm 1

```

1.  begin
2.       $rel_0 = rel(R, n)$  and  $T_0 = \emptyset$ ;
3.      do until  $rel_{i-1} = \emptyset$ 
4.           $T_i$  is the set of maximal relations of  $rel_i = rel_{i-1} - T_{i-1}$ 
              with respect to  $\sqsubseteq$ ;
5.      return Result =  $\{T_1, \dots, T_b\}$ ;
6.  end.
    
```

Let the last precision level generated by the algorithm be T_b and $\{T_1 < T_2 < \dots < T_b\}$ be a collection of subsets obtained by the above algorithm, where the linear ordering is according to the order of generation of T_i in the steps 3 and 4. Clearly, the returned result is a linearly ordered partition of $rel(R, n)$. We still need to show that Definition 7 is well-defined as in next Lemma.

Lemma 1. Algorithm 1 terminates and the generated precision hierarchy generated is unique.

Proof.

It is trivial that the above algorithm will terminate for a finite domain $\mathcal{D}$. This can be easily established by using induction on T_i . The uniqueness of the precision hierarchy follows from the fact that T_i is the unique set of all maximal tuples of rel_i with respect to $\sqsubseteq$. $\square$

We observe that the number of relations at the top level in the precision hierarchy (i.e. the number of the most precise relations) is $|T_1| = |\mathcal{D}|^{n|R|}$. For example we can verify that in Figure 3 there are $2^{2 \times 1} = 4$ relations at the precision level T_1 . We will also present in Lemma 2 some interesting properties of the precision hierarchy described in Definition 7. Let us give some intuition of this Lemma in the first place. Informally, the statement 1 asserts the uniformity of relations with respect to precision at each level, the statement 2 asserts that a relation at a higher level will not be less precise than any relation at a lower level, and finally the statement 3 asserts that at least one relation at a higher level is strictly more precise than some relations at a lower level. These three properties correspond to the requirements pertaining to the general concept of an *internal hierarchy* defined upon a partially ordered set [13]. An internal hierarchy can also be viewed as a generalization of the concept of multi-resolution, which is a well-recognized means to approximate the result of image data retrieval [8].

Lemma 2. The following statements are true for the precision hierarchy of $rel(\mathbf{R}, n)$.

1. $\forall T \in part(\mathbf{R}, n), \forall r_1, r_2 \in T$, either $r_1 = r_2$ or $r_1 \parallel r_2$ (i.e., T is unordered).
2. $\forall T_i, T_j \in part(\mathbf{R}, n), T_i < T_j \Rightarrow \forall r_1 \in T_i, \forall r_2 \in T_j, r_2 \not\sqsubseteq_1 r_1$.
3. $\forall T_i, T_j \in part(\mathbf{R}, n), T_i < T_j \Rightarrow \exists r_1 \in T_i, \exists r_2 \in T_j$ such that $r_1 \sqsubset r_2$.

Proof.

Suppose r_1 and r_2 are two distinct relations in T_i . Then we have $r_1 \parallel r_2$, since they are both the maximal relations of rel_i with respect to $\sqsubseteq$. So it satisfies the part 1. Now, we assume to the contrary that $\exists r_1 \in T_i, \exists r_2 \in T_j$ such that $r_2 \sqsubseteq r_1$ and $T_i < T_j$. Then it follows that $r_2 = r_1$, since r_1 is a maximal relation and is less than r_2 . However, this is impossible because T_i and T_j are disjoint. Hence the part 2 is also satisfied. Finally, the part 3 can be established by noting that T_i is the set of all maximal elements of some superset of T_j . It follows that for any element $r_2 \in T_j$, there is an element $r_1 \in T_i$ such that $r_1 \sqsubset r_2$. $\square$

3 Functional Dependencies in Indefinite Relations

We now formalize the notion of an FD being satisfied in an indefinite sequence relation. We note that useful references can be found in [33,5,21], in which present a comparison of various approaches in defining FDs in the context of fuzzy relations and incomplete relations.

Definition 8. (Functional Dependencies Satisfaction) A *functional dependency* (or simply an FD) over $\mathbf{R}$ is a statement of the form $X \rightarrow Y$, where $X, Y \subseteq \mathbf{R}$. An FD $X \rightarrow Y$ is satisfied in a relation r over $\mathbf{R}$, denoted by $r \models X \rightarrow Y$, if $\exists s \in poss(r)$ such that $\forall t_1, t_2 \in s$, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

The following lemma gives a syntactic characterization of the notion of satisfaction of an FD in indefinite sequence relations.

Proposition 2. $r \models X \rightarrow Y$ if and only if $\forall t_1, t_2 \in r$, if $\forall A \in X \mid t_1[A] = t_2[A] \mid = 1$ and $t_1[X] = t_2[X]$, then $\forall A \in Y, t_1[A] \cap t_2[A] \neq \emptyset$. $\square$

We first formalize the notion of *logical implication* and an *axiom system*, and then show that the famous *Armstrong's axiom system* [2] for FDs in conventional or fuzzy relations [33,21] is not sound in the context of indefinite sequence relations. On the other hand, Lien and Atzeni's axiom system is sound and complete in this case, which is also a classic example of *axiom systems* in the literature of relational database theory [38,3].

Definition 9. (Logical Implication and Axiom System) A set of functional dependencies F *logically implies* a functional dependency f over $\mathbf{R}$, written $F \models f$, whenever for all relations r over $\mathbf{R}$, if for all $f' \in F, r \models f'$ holds, then $r \models f$ also holds. An *axiom system* $\mathcal{A}$ for F is a set of inference rules (or simply

rules) that can be used to *derive* functional dependencies from F over R . We say that f is *derivable* from F by $\mathcal{A}$, if there is a finite sequence of functional dependencies over R , whose last element is f , and where each functional dependency in the said sequence is either in F or follows from a finite number of previous functional dependencies in the sequence by one of the inference rules. We denote by $F \vdash f$ the fact that f is *derivable* from F by a specified axiom system.

It is well-known that in standard relational databases [38,3], Armstrong's axiom system comprising the three inference rules of reflexivity, argumentation and transitivity is sound and complete for FDs. We now show a counter-example to the soundness of the transitivity rule for FDs in an indefinite sequence relation.

Lemma 3. The transitivity rule is unsound for FDs in indefinite sequence relations.

Proof.

We use the following counter-example to prove our claim.

$$r = \begin{array}{c|c|c|c} & A & B & C \\ \hline t_1 & \{0\} & \{0\} & \{0\} \\ t_2 & \{0\} & \{0, 1\} & \{1\} \end{array}$$

Fig. 4. A relation shows that the transitivity rule is unsound for FDs.

Consider the relation r given in Figure 4. Let $F = \{A \rightarrow B, B \rightarrow C\}$. Then it can be checked that $r \models F$ but $r \not\models A \rightarrow C$. $\square$

Lien and Atzeni's axiom system for incomplete relations is different from Armstrong's axiom for conventional FDs in the following way: it drops the transitivity rule and adds the union rule and the decomposition rule. Lien and Atzeni's axiom system will be shown to be sound and complete for FDs in indefinite sequence relations in Theorem 1. We first re-state this axiom system as follows.

Definition 10. (Lien and Atzeni's Axiom System) Let X, Y, Z be subsets of R , $A \in R$ and F be a set of FDs. *Lien and Atzeni's axiom system* constitutes the following inference rules for FDs.

- (FD1) *Reflexivity*: if $Y \subseteq X$, then $F \vdash X \rightarrow Y$.
- (FD2) *Augmentation*: if $F \vdash X \rightarrow Y$ and $W \subseteq R$, then $F \vdash XW \rightarrow YW$.
- (FD3) *Union*: if $F \vdash X \rightarrow Y$ and $F \vdash X \rightarrow Z$, then $F \vdash X \rightarrow YZ$.
- (FD4) *Decomposition*: if $F \vdash X \rightarrow Y$, then $\forall A \in X, F \vdash X \rightarrow A$.

We now show our main theorem which states that the above axiom system is sound and complete for FDs, holding in indefinite sequence relations. The underlying idea in this proof is standard [38]. We make use the notion of *closure* which is defined as follows, the *closure* of a set of attributes, $X \subseteq R$, with respect to a given set of FDs F , denoted as X^+ , is given by $X^+ = \{A \mid F \vdash X \rightarrow A\}$. We also need to assume that each domain has at least two distinct elements. This assumption is reasonable in a practical DBMSs.

Theorem 1. The axiom system comprising from FD1 to FD4 is sound and complete for FDs.

Proof.

It is easy to show that the inference rules from FD1 to FD4 are sound. We prove completeness by showing that if $F \vdash X \rightarrow Y$, then $F \not\models X \rightarrow Y$. Equivalently for the latter, it is sufficient to exhibit a relation, say r , such that $r \models F$ but $r \not\models X \rightarrow Y$. Let r be the relation consisting of two tuples t_1 and t_2 shown in Figure 5, where $Q = R - X^+$ and $Z = X^+ - X$.

	X	Q	Z
t_1	$\{1\} \cdots \{1\}$	$\{1\} \cdots \{1\}$	$\{0, 1\} \cdots \{0, 1\}$
t_2	$\{1\} \cdots \{1\}$	$\{0\} \cdots \{0\}$	$\{0, 1\} \cdots \{0, 1\}$

Fig. 5. A relation r showing that $r \not\models X \rightarrow Y$

We first show that $r \models F$. Suppose to the contrary that $r \not\models F$, and thus there exists an FD, $C \rightarrow D \in F$ such that $r \not\models C \rightarrow D$. It follows by the construction of r and Proposition 2 that $C \subseteq X$ and that $\exists A \in (D \cap Q)$ such that $A \notin X^+$. By FD2 it follows that $F \vdash X \rightarrow DX$, and by FD4, it follows that $F \vdash X \rightarrow A$. This leads to a contradiction, since it follows that $A \in X^+$. We conclude the proof by showing that $r \not\models X \rightarrow Y$. Suppose to the contrary that $r \models X \rightarrow Y$. By the construction of r and Proposition 2 again, $A \subseteq X^+$ for all $A \in Y$. This leads to a contradiction, since by FD3 it follows that $F \vdash X \rightarrow Y$. $\square$

4 Non-additivity of FDs in Indefinite Sequence Relations

Herein we discuss the additivity problem for FDs in indefinite sequence relations. It is well-known that in conventional relations the satisfaction of a set of FDs is *additive* [3], meaning that a given relation satisfies a set of FDs if and only if the relation satisfies individual FDs in the set. However, we find that in indefinite sequence relations the satisfaction is not additive, i.e. it may be the case that, given a relation r and a set of FDs F , r satisfies each FDs in F but r does not satisfy F . The lack of additivity for satisfaction gives rise to the problem that Lien and Atzeni's axiom system, which is sound and complete with respect to the satisfaction of a single FD, but in general, does not cater for the satisfaction for a set of FDs.

Definition 11. (Additive Satisfaction) Satisfaction is said to be *additive* with respect to a class of relations over R , $rel(R, n)$, and a set of FDs F , if, $\forall r \in rel(R, n)$, $r \models F$, if and only if, $\forall f \in F$, $r \models f$.

Obviously, the only if part of the above definition trivially holds. The satisfaction is said to be *non-additive* if it is not additive. We use the following example to illustrate the non-additivity of the satisfaction for FDs in indefinite sequence relations.

Example 3. Let r_1 be a relation over $\{A, B, C\}$. The relation in Figure 6 (a) shows that $r_1 \models A \rightarrow B$ and $r_1 \models B \rightarrow C$, but $r_1 \not\models \{A \rightarrow B, B \rightarrow C\}$. This is due to the fact that, although $\exists r \in \text{poss}(r_1)$ such that $r \models A \rightarrow B$ and $\exists r \in \text{poss}(r_1)$ such that $r \models B \rightarrow C$, we have $\forall r \in \text{poss}(r_1), r \not\models \{A \rightarrow B, B \rightarrow C\}$. The similar comments apply to the relation in Figure 6 (b), which shows that $r_2 \models A \rightarrow C$ and $r_2 \models B \rightarrow C$, but $r_2 \not\models \{A \rightarrow C, B \rightarrow C\}$. Finally, the relation in Figure 6 (c) shows that $r_3 \models B \rightarrow A$ and $r_2 \models AC \rightarrow B$ but $r_2 \not\models \{B \rightarrow A, AC \rightarrow B\}$.

A	B	C
{0}	{0}	{0}
{0}	{0, 1}	{1}

(a) relation r_1

A	B	C
{0}	{0, 1}	{0}
{0}	{0}	{0, 1}
{0, 1}	{0}	{1}

(b) relation r_2

A	B	C
{0}	{0}	{0, 1}
{0, 1}	{0}	{0}
{0}	{1}	{0}

(c) relation r_3

Fig. 6. An example of relations showing non-additivity

We now summarise our discussion in the following lemma. Note that for the case $\text{rel}(\mathbf{R}, 1)$, the satisfaction of $\mathbf{F}$ over $\mathbf{R}$ becomes trivial.

Lemma 4. The following statements are true.

1. Satisfaction is non-additive with respect to $\text{rel}(\mathbf{R}, n)$ and a set of $\mathbf{F}$ over $\mathbf{R}$, where $n > 1$.
2. Satisfaction is additive with respect to $\bigcup_{r \in \text{rel}(\mathbf{R}, n)} \text{poss}(r)$ and a set of $\mathbf{F}$ over $\mathbf{R}$.

Proof.

The first part follows from Definition 11 and the counterexamples given in Example 3. The second part is immediate, since Definition 11 vacuously holds for definite sequence relations. $\square$

5 Concluding Remarks

We have defined indefinite sequence relations and described how the representation of indefinite sequence information is a valuable extension of the relational database, following on from the work in [17, 39]. We also defined the partial order $\sqsubseteq$ on tuples, which corresponds to the notion of “definiteness” for indefinite sequence relations. In addition, we established a formal hierarchical structure over the class of relations having a fixed schema and cardinality in Definition 7. This hierarchy can be employed to classify the relations into different levels of precision as shown in Lemmas 1 and 2.

We extended FDs to indefinite sequence relations, using the concept of the set of possible definite sequence relations. A FD is being satisfied in an indefinite sequence relation, if there exists a possible definite sequence relation satisfying

the FD extracted from it. We showed two important results related to the satisfaction of FDs in the context of indefinite sequence relations. First, we showed in Theorem 1 that Lien and Atzeni's axiom system is sound and complete for such FDs. The theorem implies that we can apply the well-established result in incomplete results such as the closure and implication algorithms (c.f. Algorithms 3.1 and 3.2 in [3]) to FDs in indefinite sequence relations. Second, we showed that in Lemma 4 the FD satisfaction is in general non-additive. Further work can be done to characterise the class of relations that have additive satisfaction in order to make Lien and Atzeni's axiom system to be more general in proving the implication of a set of FDs.

To this end we have assumed that we have a single indefinite sequence relation r over R and a set of FD F . It would be interesting to extend the results of this paper to deal with the case of indefinite sequence database, which is a family of indefinite sequence relations (c.f. [26]). Another interesting area is to study the semantics of INDs in indefinite sequence databases, since INDs generalize the notion of referential integrity and foreign keys [6,9]. It also deserves an investigation on the impact of the result of this paper if an indefinite tuple is generalized into a tuple whose cell values are general fuzzy sets (i.e. memberships of the elements in a cell may not be identical). In this case, the degree of satisfaction of FDs in an indefinite sequence relation r is related to the membership of the consistent definite sequence relations in the possible world $poss(r)$. The informal reason is that those consistent definite sequence relations have different "weight" in $poss(r)$ and thus a threshold of the satisfaction for an FD should be considered.

Acknowledgments:

Various comments and inspiration from Ethan Collopy, Zhang Fan, Hans-Joachim Klein and the anonymous referees in different stages have been most useful.

References

1. S. Abiteboul and S. Ginsburg. Tuple Sequences and Lexicographical Indexes. *Journal of the Association for Computing Machinery* **33**(3), pp. 409-422 (1986). 400
2. W.W. Armstrong. Dependency Structures of Data Base Relationships. In *Proceedings of the IFIP Congress*, Stockholm, pp. 580-583, (1974). 406
3. P. Atzeni and V. De Antonellis. *Relational Database Theory*. Benjamin/Cummings Publishing Company, Inc., (1993). 399, 401, 401, 406, 407, 408, 410
4. B.R. Badrinath and T. Imielinski. Replication and Mobility. In *Proceedings of the 2nd IEEE Workshop on Management of Replicated Data*, (1992). 399
5. P. Bosc, D. Dubois and H. Prade. Fuzzy functional dependencies - An Overview and a Critical Discussion. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, Orlando, FL, pp. 325-330, (1994). 401, 406
6. E.F. Codd. Extending Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, **4**, pp. 397-434, (1979). 410

7. W. Feller. *An Introduction to Probability Theory and Its Applicaitons*. John Wiley & Sons Publishing Company, Inc., 2nd edition, (1971). 401
8. R. Read. *Towards Multiresolution Data Retrieval via the Sandbag*. Ph.D. Thesis, University of Texas at Austin, United States, (1995). 405
9. D.S. Johnson and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences*, **28**, pp. 167-189, (1984). 410
10. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York, (1979).
11. S. Ginsburg and K. Tanaka. Computation-Tuple Sequences and Object Histories. *ACM Transactions on Database Systems* **11**(2), pp. 186-212, (1986).
12. G. Grahne. Dependency Satisfaction in Databases with Incomplete Information. In *Proceedings of the International Conference on Very Large Data Bases*, Singapore, pp. 37-45, (1984). 401
13. G. Gratzner. *General Lattice Theory*. NewYork: Academic Press, (1978). 405
14. R.H. Gutting, R. Zicari and D.M. Choy. An Algebra for Structured Office Documents. *ACM Transactions on Office Information Systems* **7**(4), pp. 123-157, (1989).
15. P. Halmos. *Naive Set Theory*, Springer-Verlag, New York, (1974). 402
16. P. Honeyman. Testing Satisfaction of Functional Dependencies. *Journal of the ACM* **29**, pp. 668-677, (1982). 401
17. T. Imielinski, R. Van Der Meyden and K. Vadaparty. Complexity Tailored Design: A New Design Methodology for Databases with Incomplete Information. *Journal of Computer and System Sciences*, **51**, pp.405-432, (1995). 409
18. W.H. Inmon. *Building the Data Warehouse*. John Wiley & Sons. Inc., (1996) 400
19. H-J. Klein. Efficient Algorithms for Approximating Answers to Queries Against Incomplete Relational Databases, In: *Proceedings of KRDB'99, Linköping, E. Franconi, M. Kifer (Eds.)*, pp. 26-30, (1999).
20. P. Lech and S. Andrej. *Proceedings: Rough sets and current trends in computing : first International Conference, RSCTC '98, LNCS Vol. 1424*, Warsaw, Poland, (1998). 401
21. M. Levene and G. Loizou. Maintaining consistency of imprecise relations. *The Computer Journal* **39**, pp. 114-123, (1996). 401, 406, 406
22. L. Libkin. *Aspects of Partial Information in Databases*. Ph.D. Thesis, University of Pennsylvania, United States, (1996). 399
23. Y. E. Lien. On the Equivalence of Data Models. *Journal of the ACM* **2**, pp. 333-362, (1982). 399
24. N.A. Lorentzos. DBMS Support for Time and Totally Ordered Compound Data Types. *Information Systems* **17**(5), pp. 347-358, (1992). 399, 403
25. D. Maier and B. Vance. A Call to Order, In *ACM symp. on Principles of Databases Systems*, pp. 1-16, (1993). 399
26. H. Mannila and K-J Raiha. Generating Armstrong Databases for Sets of Functional and Inclusion Dependencies. *Research Report A-1988-7*, University of Tampere, Finland, (1988). 410
27. MIM User Manual. *Logical Information Machines*, (1994). 399
28. K. Ng and R. Muntz. Parallelizing User-Defined Functions in Distributed Object-Relational DBMS. *IEEE Proceedings of the International Database Engineering and Applications Symposium*, Montreal, Canada, pp. 279-287, (1999). 400
29. W. Ng and M. Levene. The Development of Ordered SQL Packages to Support Data Warehousing. *Proceedings of the 8th International Database Workshop*, Hong Kong, pp. 208-235, (1997). 403

30. W. Ng and M. Levene. The Development of Ordered SQL Packages for Modelling Advanced Applications. *Lecture Notes in Computer Science Vol. 1308: Database and Expert Systems Applications*, Toulouse, France, pp. 529-538, (1997). 400
31. W. Ng. Lexicographically Ordered Functional Dependencies and Their Application to Temporal Relations. *IEEE Proceedings of the International Database Engineering and Applications Symposium*, Montreal, Canada, pp. 279-287, (1999). 399, 401
32. W. Ng. Functional Dependencies in Ordered Relational Databases. To appear in *Information Systems*, (1999). 399, 401
33. K.V.S.V.N. Raju and A.K. Majumdar Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems. *ACM Transactions on Database Systems* **13**(2), pp. 129-166, (1988). 401, 401, 406, 406
34. P. Seshadri, M. Livny and R. Ramakrishnan. The Design and Implementation of a Sequence Database System. *Proceedings of the 22nd VLDB Conference*, Mumbai, India, pp. 99-110, (1996). 399, 399
35. S. Sheno, A. Melton and L.T. Fan. An Equivalence Classes Model of Fuzzy Relational Databases. *Fuzzy Sets and Systems*, **38**, pp. 153-170, (1990). 400, 401, 401
36. A. Silberschatz, M. Stonebraker and J.D. Ullman. Database Systems: Achievements and Opportunities. *Sigmod record*, **19**(4), pp 6-22, (1990). 399
37. A. Tansel et al. (editors). *Temporal Databases: Theory, Design and Implementation*. The Benjamin/Cummings Publishing Company, Inc., (1993) 403
38. J.D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I*, Rockville, MD., Computer Science Press, (1988). 401, 401, 406, 407, 407
39. K. Vadaparty and S. Naqvi. Using Constraints for Efficient Query Processing in Nondeterministic Databases. *IEEE Transactions on Knowledge and Data Engineering* **7**, pp.850-864, (1995). 409

View Integration of Object Life-Cycles in Object-Oriented Design^{*}

Günter Preuner¹ and Stefan Conrad²

¹ Department of Business Information Systems
University of Linz, A-4040 Linz, Austria
`preuner@dke.uni-linz.ac.at`

² Institute of Technical and Business Information Systems
University of Magdeburg, D-39106 Magdeburg, Germany
`conrad@iti.cs.uni-magdeburg.de`

Abstract. Object-oriented database schemas are often defined by several future users of the planned database, where each user defines a schema representing his/her view on the database; the complete database schema is defined by integrating these views in a subsequent step.

Behavior of objects is often defined at two levels of detail: at the level of methods and at the level of object life-cycles that represent the overall behavior of objects during their life time.

This paper presents an approach to integrate views of object life-cycles to a complete object life-cycle of an object type. We will particularly consider the problem that different users know about parts of the object life-cycle at different levels of detail and often do not consider exactly the same set of entities.

1 Introduction

Object-oriented database schemas are usually defined in cooperation with potential future users of a database. These users know the entities that should be represented by objects in the database from their everyday work and thus are able to define the structure and behavior of object types. Methods describe single activities performed on an object whereas object life-cycles represent the overall behavior of objects during their life time. In this paper we will concentrate on the definition and integration of object life-cycles.

Users are usually able to define in detail those parts of behavior that they know from their everyday work whereas they know about activities that are performed by other users only roughly or even not at all. Correspondingly each view schema defines some information in detail, includes some behavior only at an abstract level, and misses some parts of behavior. During view integration, these user views are collected and integrated by including the most specific information from each view.

^{*} This work was partly supported by the EU under ESPRIT-IV WG 22704 ASPIRE (Advanced modeling and SPecification of distributed InfoRmation systEms).

Consider, e.g., two views of an object type for room reservations in a hotel. One view is defined by the reservation department that handles requests for rooms, whereas the other view is defined by the reception. The first view contains information about the activities performed for reserving a room in detail, but knows about the use of the room only abstractly. The second view considers the use of a room with check-in, check-out, and payment in detail, but does not know details about the preceding step of booking. Further, the reservation department has to consider requests that must be declined because the hotel is booked out for the time in question. Those requests are not visible to the reception. The reception, however, treats requests of guests that come to the reception and ask for a room without having reserved a room before.

In this paper we will identify two main differences between views of the object life-cycle of an object type: (1) heterogeneities that arise between two or more views because different users perceive activities at different levels of detail; (2) heterogeneities that arise because some users deal with entities that are not visible to some other users. Based on these heterogeneities we will introduce an approach for view integration.

We will use *Object/Behavior diagrams (OBD)* [11] where object life-cycles are defined by behavior diagrams that are based on Petri nets. OBD was extended for the modeling of business processes in [2]. Specialization of object life-cycles from one object type to a subtype was treated in detail in [18,19]. Although we present our approach using behavior diagrams, the main results can also be applied to models based on statecharts [10] (e.g., UML [3]).

The problem of schema integration is well known from design methods that use view integration and from federated database systems (cf., e.g., [1,21,4]). Most approaches that have been published so far treat integration of *structure* of object types. Intensional and extensional aspects are distinguished in [17]. Recently, *description logics* have been used for representing object-oriented schemas and for their integration (cf., e.g., [13] for a discussion). Some approaches consider the integration of methods (e.g., [22,23]) and integrity constraints (e.g., [5]), but little work has been done so far on integration of object life-cycles. In [6], the authors introduced *conflict-freeness* to determine whether dynamic schemas can be integrated. Engels et al. [8] define integration of views based on graph transformations. The authors of [9] use statecharts for view integration. They introduced five different ways to connect views of object life-cycles depending on how objects have to pass the object life-cycles of the views. View integration of behavior is briefly discussed for *state nets* in [7].

In earlier works [16,14], we discussed integration of views of object life-cycles if the views are defined based on the same extension, which means that all users considered the same set of entities when defining their view schemas. In this work, we will extend this first approach to deal with the situation that different users define views of the same object type but consider different, but possibly overlapping sets of objects. In the example given before, the views define behavior of overlapping sets of reservations as both views treat reservations, that are booked first and used later; yet, each view deals with objects that are not visible

to the other view, e.g., requests that are refused by the reservation department and rooms that are used without having reserved them before. This extended approach may be better applicable in practice; yet, we will find out that the extended approach uses the basic concepts of the approach introduced in [16].

A different problem was handled in [15], where business processes of different enterprises were integrated to define a common process which contains all common parts of the original processes. This problem deviates from view integration as *existing* processes have to be observed at a common level of detail.

The remainder of this paper is structured as follows: In Sect. 2, we will introduce *behavior diagrams* as far as necessary for this paper; Sect. 3 motivates the problem of integration of behavior diagrams and presents an overview of the integration process. The main topic of this paper is the integration of views of object life-cycles with disjoint extensions; the correctness criteria and the integration steps are presented in Sect. 4. Finally, Sect. 5 concludes this work.

2 Behavior Diagrams

In this section, we briefly introduce *Behavior Diagrams* with arc labels and their specialization. For more details, the reader is referred to [18,19].

2.1 Labeled Behavior Diagrams

Behavior diagrams are based on Petri nets and consist of activities, states, and arcs. Activities correspond to transitions in Petri nets and represent work performed with business objects, states correspond to places in Petri nets and show where an object currently resides. Each instance of an object type is represented by a unique token, the object identifier, which may reside in one or several states. An activity may be invoked on an object if all prestates are marked by this object. When the execution is completed the object is inserted into all poststates of the activity. In difference to Petri nets, we assume that activities may take some time. During the execution of an activity on some object, the object resides in an implicit activity state named after the activity.

Example 1. Fig. 1 shows a behavior diagram of object type RESVT. Activities are depicted by rectangles, and states are depicted by rectangles with a trapezium at the top and the bottom. Activity *issue* creates a new reservation object. After completion of this activity, the object resides in states *toCheckIn* and *toPay*, where activities *use* and *payByCheque* or *payCash* may be started. The virtual state α will be explained later in this section.

Behavior diagrams may be *labeled*. The idea of labeling is taken from the processing of business objects by paper work where different actors work on different carbon copies of a business form. In this analogy, a label corresponds to a particular carbon copy. The labels of an arc (state, or activity) indicate which copies of a form flow along an arc, (reside in some state, or are processed by an activity, resp.). Notice, however, that *there exist no actual copies of a*

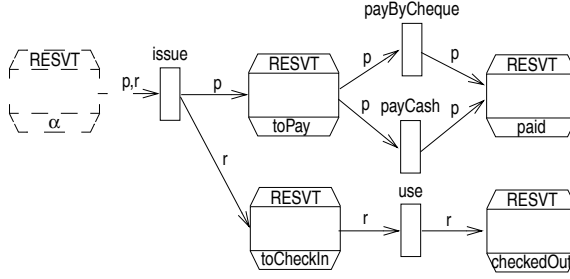


Fig. 1. Behavior diagram of RESVT

business object, but several activities and states may refer at the same time to the same business object by its object identifier. Labels have been introduced in [19] to facilitate the check whether a behavior diagram constitutes a consistent refinement of another behavior diagram.

Example 2. The business process shown in Fig. 1 uses two labels p and r corresponding to the payment and the registration copy of the reservation form.

Definition 1. A labeled behavior diagram (LBD) $B = (S, T, F, L, l)$ of an object type consists of a set of states $S \neq \emptyset$, a set of activities $T \neq \emptyset$, $S \cap T = \emptyset$, a set of arcs $F \subseteq (S \times T) \cup (T \times S)$, and a set of labels $L \neq \emptyset$. The labeling function $l : F \rightarrow 2^L \setminus \{\emptyset\}$ (2^X denotes the power set of X) assigns a non-empty set of labels to each arc in F . States, activities, and labels are referred to as elements. There is exactly one initial state $\alpha \in S$ such that $\nexists t \in T : (t, \alpha) \in F$.

States and activities are labeled, too, where the set of labels of a state or activity is the union of the sets of labels of its incident arcs. The initial state α is labeled with all labels and represents the period of time when a business object has not yet been created; it is usually omitted in the graphical representation.

At any time, an object resides in a non-empty set of states, its *life-cycle state*.

Definition 2. A life-cycle state (LCS) σ of an object is a subset of $(S \cup T) \times L$. The initial LCS is $\sigma_A = \{(\alpha, x) | x \in L\}$.

As the execution of activities may take some time and an object resides in an activity state during execution, we distinguish *start* and *completion* of an activity. An activity t can be *started* on an object if all of the activity's prestates are marked with the object; starting t yields a new life-cycle state which contains activity state t but no prestates of t . Activity t can be *completed* on an object if the object resides in activity state t ; completion yields a new life-cycle state that contains all poststates of t , but does not include t any more.

A *life-cycle occurrence* is defined as the sequence of life-cycle states of a certain object:

Definition 3. A life-cycle occurrence (LCO) γ of an object is a sequence of LCSs $\gamma = [\sigma_1, \dots, \sigma_n]$, such that $\sigma_1 = \sigma_A$, and for $i = 1, \dots, n-1$, either $\sigma_i = \sigma_{i+1}$, or there exists an activity $t \in T$ such that either t can be started on σ_i and the start of t yields σ_{i+1} or σ_i contains t and the completion of t yields σ_{i+1} .

Example 3. Consider the behavior diagram shown in Fig. 1. A possible life-cycle occurrence of a reservation object is $[[\{(\alpha, p), (\alpha, r)\}, \{(\text{issue}, p), (\text{issue}, r)\}, \{(\text{toPay}, p), (\text{toCheckIn}, r)\}, \{(\text{toPay}, p), (\text{use}, r)\}, \{(\text{toPay}, p), (\text{checkedOut}, r)\}, \{(\text{payCash}, p), (\text{checkedOut}, r)\}, \{(\text{paid}, p), (\text{checkedOut}, r)\}]$.

2.2 Specialization of Behavior Diagrams

The behavior diagram of an object type may be specialized in a subtype in two ways: by *refinement*, i.e., by decomposing states and activities into subdiagrams and labels into sublabels, or by *extension*, i.e., by adding states, activities, and labels. *Observation consistency* as a correctness criterion for specialization guarantees that any life-cycle occurrence of a subtype is observable as a life-cycle occurrence of the supertype if extended elements are ignored and refined elements are considered unrefined. Observation consistency allows “*parallel extension*” but not “*alternative extension*”, i.e., an activity that is added in the behavior diagram of a subtype by extension may not consume from or produce into a state that the subtype inherits from the behavior diagram of the supertype. Observation consistency requires only *partial inheritance* of activities and states: “alternatives” modeled in the behavior diagram of a supertype may be omitted in the behavior diagram of a subtype (cf. [18]¹).

We use a total specialization function $h : S' \cup T' \cup L' \rightarrow S \cup T \cup L \cup \{\varepsilon\}$ to represent the correspondences between a more special behavior diagram $B' = (S', T', F', L', l')$ and a behavior diagram $B = (S, T, F, L, l)$. The inheritance function h can express four cases, (1) inheritance without change, (2) refinement, (3) extension, and (4) elimination, as follows: (1) If an element e is not changed, then $\exists e' \in S' \cup T' \cup L' : h(e') = e$ and $\forall e'' \in S' \cup T' \cup L', e'' \neq e' : h(e'') \neq h(e')$. (2) If an element e in B is refined to a set of elements E ($|E| > 1$), then $\forall e' \in E : h(e') = e$. (3) If a set of elements E is added in B' , then $\forall e \in E : h(e) = \varepsilon$. (4) If a set of states and activities $E \subseteq S \cup T$ is removed in B' , then $\forall e \in E \nexists e' \in S' \cup T' \cup L' : h(e') = e$.

For the definition of *observation consistent specialization*, we need to define the *generalization of a life-cycle state* and the *generalization of a life-cycle occurrence*.

¹ Omitting states and activities of a supertype in a subtype seems to be counter-intuitive at first. Yet, a more in depth analysis reveals that partial inheritance is coherent with a co-variant specialization of method preconditions followed in conceptual specification languages.

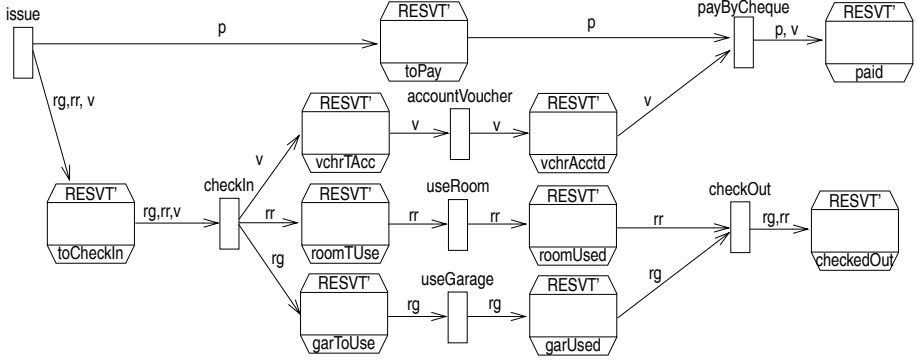


Fig. 2. Business process of RESVT'

Definition 4. A generalization of a life-cycle state σ' of an LBD B' of object type O' to object type O with LBD B , denoted as σ'/O , is defined as $\sigma'/O \subseteq (S \cup T) \times L$, where $\forall e \in S \cup T, x \in L : ((e, x) \in \sigma'/O \Leftrightarrow \exists e' \in S' \cup T', x' \in L' : h(e') = e \wedge h(x') = x \wedge (e', x') \in \sigma')$.

Definition 5. A generalization of a life-cycle occurrence $\gamma' = [\sigma'_1, \dots, \sigma'_n]$ of an LBD B' of object type O' to object type O is defined as $\gamma'/O = [\sigma'_1/O, \dots, \sigma'_n/O]$.

Definition 6. An LBD B' of object type O' is an observation consistent specialization of an LBD B of object type O if and only if for any possible LCO γ' of B' holds: γ'/O is an LCO of B .

Example 4. The behavior diagram of object type RESVT' shown in Fig. 2 is an observation consistent specialization of the behavior diagram of object type RESVT in Fig. 1. Activity use has been refined to a subnet consisting of activities checkIn, useRoom, useGarage, and checkOut and several states between these activities. Label r has been decomposed into sublabels rr (registration for a room) and rg (registration for a parking lot in the garage). Activity accountVoucher, states vchrTAcc and vchrAcctd, and label v have been added by extension. Activity payCash has been omitted by extension, which requires only partial inheritance.

A set of sufficient and necessary rules has been introduced in [19] to check for *observation consistent refinement* and *extension* of LBDs. These rules can be applied to check for *observation consistent specialization* of LBDs, too, if specialization is considered as a concatenation of refinement and extension.

Definition 7. An LBD B is an observation consistent generalization (restriction, abstraction, respectively) of B' if and only if B' is an observation consistent specialization (extension, refinement, respectively) of B .

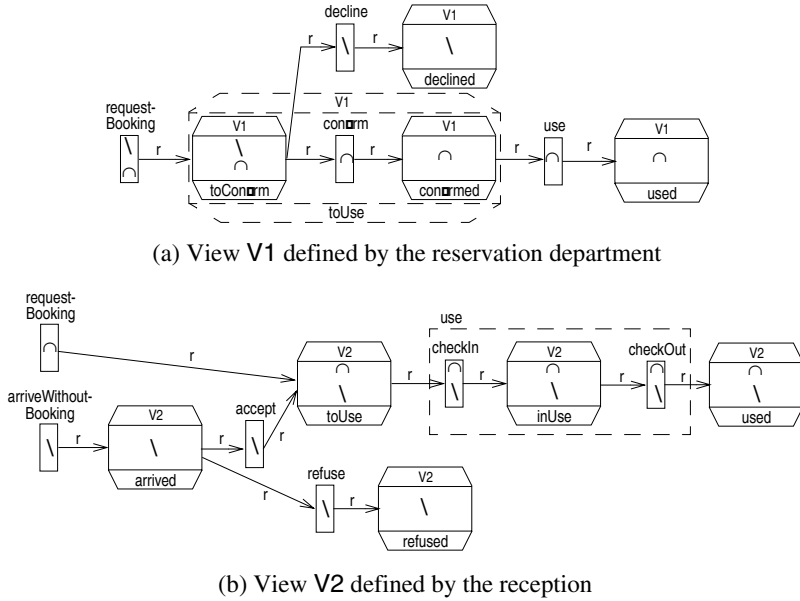


Fig. 3. Views defined by different departments

3 Overview of View Integration

In this section, we will give an overview of the problems arising during view integration. To motivate the problem of integration and the heterogeneities that may arise, we discuss a small example in Sect. 3.1; based on this example, we will identify correspondences and heterogeneities between the object life-cycles to integrate in Sect. 3.2. In Sect. 3.3, we will present an approach on how to integrate object life-cycles such that the heterogeneities are resolved. The integration process is treated in more detail in Sect. 4.

3.1 Motivating Example

We will motivate the problem of integrating object life-cycles using a small example. There are two views of an object life-cycle for an object type RES whose instances are room reservations. The first view (indicated as V_1) is defined by the reservation group that treats requests for room reservations, whereas the second view (V_2) is defined by the reception, which has to deal with the use of a room (that may have been reserved before) as well as with requests of guests that ask for a free room although they have not reserved one before. The views are shown in part (a) and (b) of Fig. 3, respectively. Please ignore the symbols $\setminus$ and $\cap$, which are depicted within the activity and state symbols for the moment.

There are two main kinds of differences between the views:

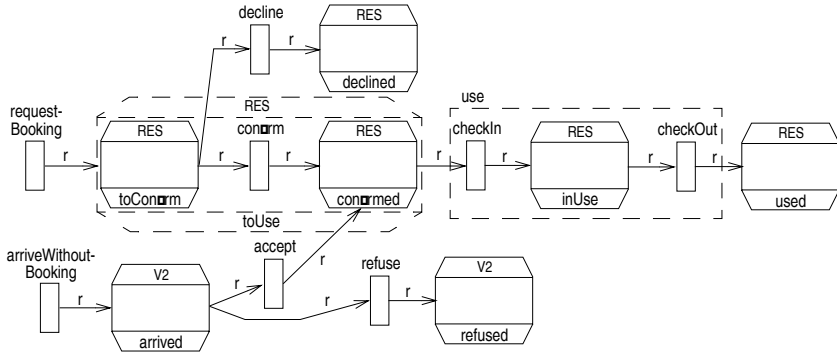


Fig. 4. Integrated object life-cycle for RES

Some activities and states may be defined in detail in one view but may be defined only as abstract elements in the other view. The reason may be that every view schema is more detailed for those activities and states that represent the everyday work of the modeler of the respective view. In the example, confirmation of a request is defined in detail in V_1 , whereas it is referred to only as an abstract state in V_2 . Similarly, activity *use* is abstract in V_1 but defined in detail in V_2 .

Some activities and states are defined in one view but not in the other one. View V_1 comprises activity *decline* and state *declined*, which are not visible in V_2 . Further, activities *arriveWithoutBooking*, *accept*, and *refuse* as well as states *arrived* and *refused* are defined only in V_2 . The reason for missing activities and states is that objects for which these activities are invoked and that reside in those states are not visible for the designer of the view since he/she is not aware of the entities that are processed according to these activities and states.

The integrated object life-cycle should define the behavior of all objects that have been considered in at least one of the views in full detail. In our example, the result of integration is shown in Fig. 4.

3.2 Correspondences and Heterogeneities

Integration of views is based on corresponding elements, i.e., activities, states, and labels that exist in both views. But as the views are defined by different users there are several heterogeneities to resolve. In this section, we will give an overview of the kinds of correspondences and heterogeneities that may arise between views. Some of them have already been motivated in the example before.

Correspondences. Elements in one view may correspond to elements in the other view in different ways: (1) *Equivalence*: A single activity, state, or label in one view is equivalent to a single activity, state, or label, respectively, in the other view if both elements have the same real-world semantics. (2) *Inclusion*: A

single activity or state in one view corresponds to a subnet consisting of several activities, states, and arcs in the other view or one label corresponds to a set of labels. (3) *Subnet correspondence*: Two subnets consisting of activities, states, and arcs correspond to each other or two sets of labels correspond to each other. (4) *No correspondence*: An element in one view does not correspond to any element in the other view.

Example 5. In our motivating example, some activities, states, and labels are equivalent to each other (e.g., `requestBooking`, `used`, and label `r`). For simplicity, they carry the same name. Activity `use` in V_1 corresponds to the subnet consisting of `checkIn`, `inUse`, and `checkOut` in V_2 by inclusion. There are no subnet correspondences in this example. Some elements have no correspondence to elements in the other view, e.g., activity `decline` and state `declined` in V_1 .

We will refer to elements that correspond to elements in the other view as *common elements* and to all other elements as *unique elements*.

Heterogeneities. Heterogeneities have been discussed in detail in the realm of integration of structure (cf., e.g., [12,20]). For behavior, we distinguish two main kinds of heterogeneities: (1) *extensional heterogeneities* that result from the fact that the designers of two views consider different extensions, i.e., different sets of entities, when defining their views; (2) *intensional heterogeneities* that arise because different activities, states, and labels are defined in the views, or elements are defined in different detail.

Extensional Heterogeneities. Basically there is one extensional heterogeneity, i.e., *missing entities*, which means that one view considers entities that are not considered in the other view.

Intensional Heterogeneities. We distinguish intensional heterogeneities as follows: (1) *Naming conflicts*: Two equivalent elements have different names (synonyms) or different elements have the same name (homonyms). (2) *Granularity conflicts*: Corresponding elements are defined in different detail, which results in an inclusion or a subnet correspondence. (3) *View-specific alternatives*: An alternative branch is defined in only one view. (4) *View-specific extensions*: View-specific extensions are unique activities and states that are labeled only with unique labels. As discussed in Sect. 2, a label introduces a *parallel* extension, i.e., a branch that is in parallel to the other branches of the object life-cycles, which carry other labels.

Example 6. For better readability, there are no naming conflicts in our example. Granularity conflicts result from the detected inclusions (e.g., state `toUse` and activity `use` and their corresponding refinements). In the example view V_2 (cf. Fig. 3), activities `arriveWithoutBooking`, `accept`, and `refuse` as well as states `arrived` and `refused` constitute a view-specific alternative.

There is no view-specific extension in our motivating example for simplicity. Yet, suppose that view V_2 contains another label `p` as well as activity `pay` with

prestate `toPay` and poststate `paid`. The reason for this view-specific extension may be that only the receptionist considers payment because the receptionist must cash the money from the guest.

3.3 Overview of the Integration Process

Due to external heterogeneities, the integration process has to distinguish the following cases. The views will be referred to as V_1 and V_2 , the extension considered in the views will be referred to as E_1 and E_2 , respectively.

Same Extension. If both views consider the same extension, they are integrated by specialization, i.e., the integrated schema is an *observation consistent specialization* of V_1 and V_2 . Then, the views can *observe* processing of objects in the integrated schema. There is no extensional heterogeneity to consider; the intensional heterogeneities are treated as follows: Granularity conflicts are resolved by integrating elements at the most detailed level, view-specific extensions are integrated, and view-specific alternatives are omitted (cf. [16,14] for details).

Disjoint Extension. The schemas V_1 and V_2 may define views of the same object type but consider disjoint extensions. They will, however, include corresponding elements, e.g., activities that are executed for all objects of E_1 and E_2 . We require that the integrated object life-cycle V defines behavior for the extension $E_1 \cup E_2$. The correctness criteria that must hold for the integrated schema and the process how to integrate such views will be discussed in Sect. 4 in detail. Intuitively, all life-cycle occurrences that are allowed in at least one of the views must be reflected in the integrated object life-cycle. Thus, all different refinements of an abstract element, all view-specific alternatives, and all view-specific extensions must be included. As V defines behavior for $E_1 \cup E_2$, i.e., for a *superclass* of E_1 and of E_2 , it must be an *observation consistent generalization* of V_1 and V_2 . Although exactly *two* object life-cycles are integrated at a time, n object life-cycles can be integrated by applying a *binary integration strategy* (cf. [1]).

Overlapping Extension. Most generally, E_1 and E_2 overlap. This case can be reduced to the cases of *same* and *disjoint* extensions (cf. Fig. 5): During *separation*, the *disjoint sub-view* $V_i^\setminus$ and the *common sub-view* $V_i^\cap$ are defined from V_i . Sub-views $V_i^\setminus$ and $V_i^\cap$ define the behavior of those subsets of extension E_i that are only considered in V_i or in both views V_1 and V_2 , respectively.

During separation, the system integrator has to determine for each sub-view which alternative branches apply to its extension. Alternative branches in V_i that must not be executed on the members of a sub-view's extension are omitted in this sub-view. The sub-views $V_i^\setminus$ and $V_i^\cap$ omit alternative branches from V_i , but they do not further change behavior. As a result, every sub-view $V_i^\setminus$ and $V_i^\cap$ is an *observation consistent specialization* of V_i , as alternatives can be omitted according to *observation consistent extension*.

Since the sub-views $V_1^\cap$ and $V_2^\cap$ are defined for the same extension, they can be *integrated by specialization* (see above), yielding the *integrated common*

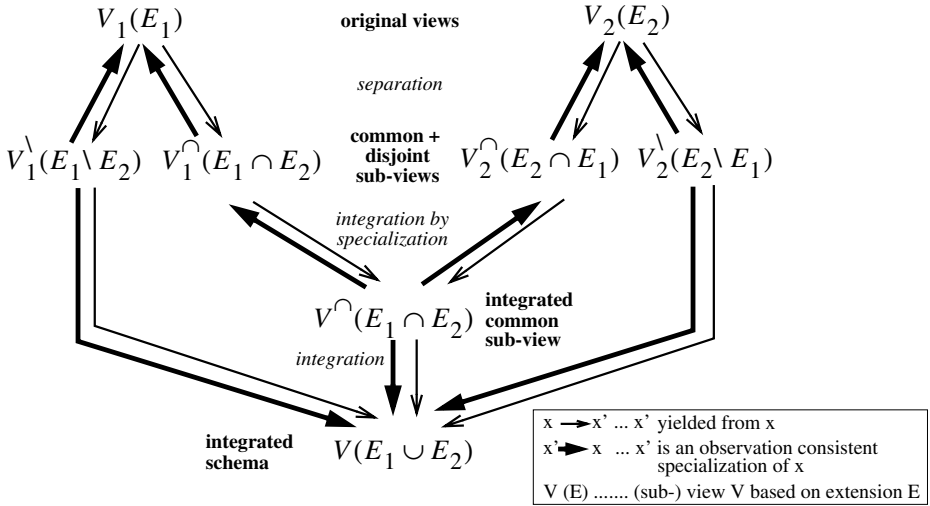


Fig. 5. Overview of integration

sub-view $V^\cap$. The sub-views $V^\cap$, $V_1^\setminus$, and $V_2^\setminus$ consider disjoint extensions such that they can be integrated according to the case *disjoint extension*; the result is the integrated schema V . In general, $2^n - 1$ sub-views must be defined for n given views.

Included Extension. As a special case of overlapping extensions, either $E_1 \setminus E_2$ or $E_2 \setminus E_1$ is the empty set, such that either $V_1^\setminus$ or $V_2^\setminus$ is empty.

Example 7. The views in the example introduced in Sect. 3.1 consider overlapping extensions: Reservations that are first reserved and then used are treated in both views, whereas declined reservations are only considered in V_1 , and arrivals without booking are only considered in V_2 . In Fig. 3, the elements defined in $V_i^\setminus$ and $V_i^\cap$ are indicated by the symbols $\setminus$ and $\cap$, respectively.

4 Integration Process

We present the integration process for the integration of (sub-) views with disjoint extensions in detail in this section. We will first define the correctness criteria for the integrated schema and then introduce the process of integration.

4.1 Correctness Criteria

The integrated object life-cycle B must be defined from the views B_1 and B_2 by adhering to the following rules:

Observation Consistent Generalization. There must be a specialization function $h_i : S_i \cup T_i \cup L_i \rightarrow S \cup T \cup L \cup \{\varepsilon\}$ (for $i \in \{1, 2\}$) such that object life-cycle $B = (S, T, F, L, l)$ is an observation consistent generalization of $B_1 = (S_1, T_1, F_1, L_1, l_1)$ and $B_2 = (S_2, T_2, F_2, L_2, l_2)$ with specialization function h_1 and h_2 , respectively.

No Loss of Information. Since all objects are treated according to the integrated schema, object life-cycle B must comprise all activities, states, and labels of B_1 and B_2 , i.e., there must not be any abstraction or restriction of elements. Thus, for the specialization function h_i it must hold that $\forall e \in S_i \cup T_i \cup L_i : h_i(e) \neq \varepsilon$ (no restriction), $\forall e' \in S_i \cup T_i \cup L_i, e'' \in S_i \cup T_i \cup L_i : h_i(e') = h_i(e'') \Rightarrow e' = e''$ (no abstraction). The only possible generalization is adding alternative branches.

Correspondence Compliance. We require that corresponding elements E_1 in B_1 and e_2 in B_2 are represented by the same element e in B , i.e., $e = h_1(e_1) = h_2(e_2)$. Since no abstractions are allowed in B , this constraint can only hold if E_1 and e_2 are equivalent (cf. Sect. 3.2). Inclusions or subnet correspondences are in fact different alternative implementations of an abstract activity or state such that they must be integrated as branches that are alternative to each other.

Correspondence of Life-Cycle Occurrences. We require that the integrated object life-cycle B reflects exactly the union of all possible life-cycle occurrences in B_1 and B_2 . Thus, we require that for any possible life-cycle occurrence γ in B , there is at least one B_i with a life-cycle occurrence γ' , such that the generalization of γ' is equal to γ (where generalization corresponds to a 1 : 1 mapping of elements). The inverse condition, i.e., any life-cycle occurrence in B_i must be valid in B , too, has been enforced by the criterion of observation consistent generalization.

4.2 Integration Steps

We will introduce three integration steps that are ideally executed sequentially. Yet in practice, some iterations may be necessary until a suitable integration can be found: (1) *Schema conformation*: The view schemas have to be transformed yielding the *conformed view schemas* B_1 and B_2 which can be integrated in a subsequent step. (2) *Determination of correspondences*: The modeler must determine corresponding elements that will be integrated to the same element. (3) *Integration*: The integrated schema B is defined from the conformed schemas based on the determined correspondences.

Example 8. The integration process will be illustrated by an example, in which three views on object types RES have to be integrated (cf. Fig. 6). Notice that we have extended our previous motivating example here to be able to illustrate some further issues of integration.

Schema Conformation. The original view schemas can be prepared for integration during schema conformation. In this step, the view schemas are not changed substantially but only restructured.

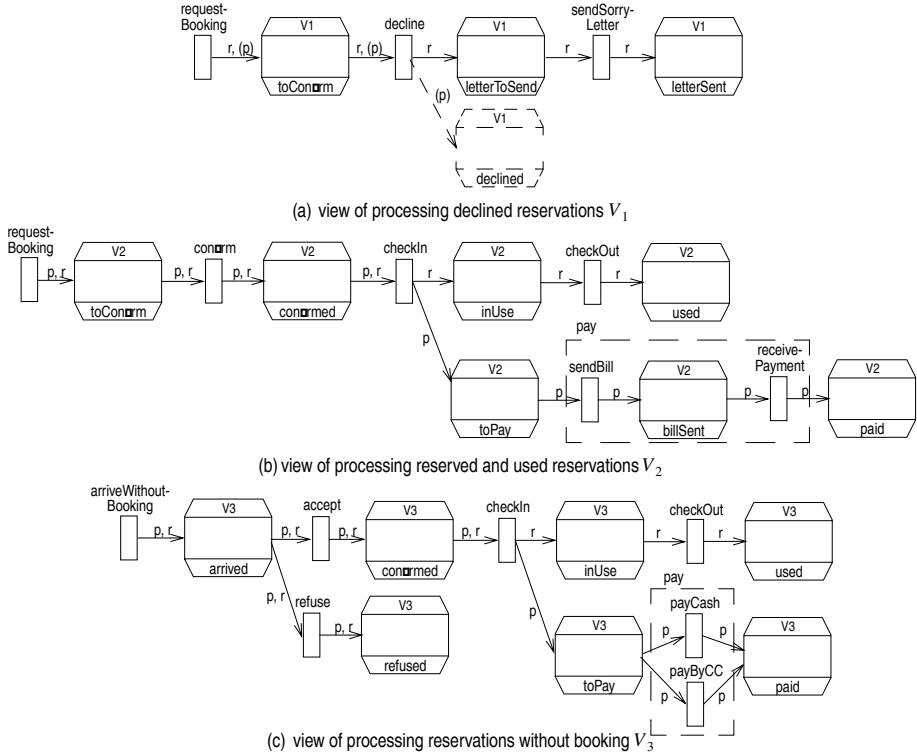


Fig. 6. Views to integrate

We assume that the original view schemas defined by the users are not completely labeled. The reason is that labels and the assignment of labels to arcs may only be determined by comparing the view schemas and thus figuring out the essential aspects considered in the views.

Adding labels is necessary to resolve the heterogeneity *view-specific extensions*, which means that a view schema comprises labels that are not defined in the other view schema. The integrated schema can only be correctly integrated if all views comprise the same set of labels. Otherwise labels that are defined in one view schema but are missing in the other one would have to be omitted during integration to fulfill the condition of *observation consistent generalization*. Yet, omitting labels during integration would violate the rule *no loss of information*.

If labels are added to a view schema, it may be necessary to add *non-substantial* states, i.e., states that do not further restrict the possible sequences of activity invocations. Non-substantial states that are labeled with a set of labels are useful to represent the fact that these labels reside in some “waiting state” during processing.

Example 9. In our example, label **p** (representing the aspect of payment) and state **declined** are introduced in view V_1 during schema conformation.

As a correctness criterion, we require that the possible sequences of activity invocations must be the same in the original behavior diagram and in the conformed behavior diagram.

Intuitively, we may compare additional labels with additional copies of a paper form in paper work: If a user needs an additional copy of a paper form for some reason, this copy is added to the paper form. Another user will not define this copy unless he/she needs it, too. Yet during integration, that set of paper forms is introduced that fulfills the requirements of all users.

Determination of Correspondences. The modeler has to determine correspondences between the view schemas. Thereby, the following restrictions apply:

1. *Correspondence of equivalent elements:* An activity (state, or label) in B_1 can be considered *equivalent* to another activity (state, or label, respectively) in B_2 only if both elements represent the view of the same element in the integrated schema (cf. the correctness criterion of *correspondence compliance*).
2. *Equivalence of labels:* Since no restriction or abstraction is allowed during integration, the set of labels in B_1 and B_2 must be equivalent, i.e., for each label in B_1 , there must be an equivalent label in B_2 and vice versa. If this condition is violated, the modeler has to insert new labels during schema conformation.
3. *Inclusions and subnet correspondences:* Inclusions and subnet correspondences exist if the view schemas provide different refinements of an abstract activity or state.
4. *Disjointness of correspondences:* An element in one view is either unique (i.e., it does not correspond to any element in the other view) or it corresponds to exactly one element or subnet in the other view.

Example 10. In Fig. 6, equivalent elements carry the same name for better readability (e.g., activity **requestBooking** in V_1 and V_2 , state **inUse** in V_2 and V_3 , label **p** in all views). Views V_2 and V_3 define different corresponding subnets for an abstract activity **pay**. In this example, we assume that room reservations that have been reserved in the reservation department (cf. V_2) can be paid after a bill has been sent to the customer; reservations that have not been reserved in advance must be paid during the stay in the hotel by cash or credit card. There are unique elements, too (e.g., activity **decline** in V_1 , state **refused** in V_3).

Integration. The integrated object life-cycle B is constructed from B_1 and B_2 based on the set of correspondences as follows:

1. *Equivalences:* One element is defined for a pair of equivalent elements in the views. *Naming conflicts* can be resolved by renaming.

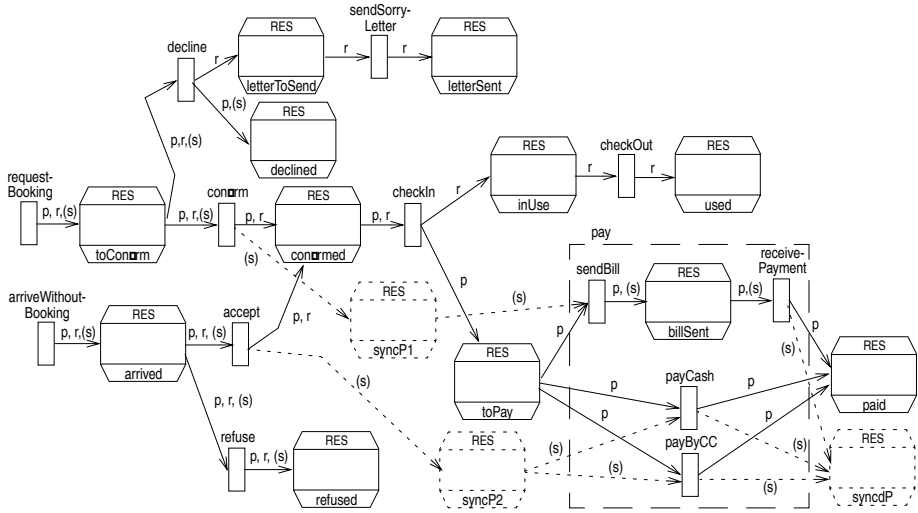


Fig. 7. Integrated object life-cycle for object type RES

2. *Inclusions and subnet correspondences*: Elements that belong to inclusions or set correspondences are defined in B as they are integrated as branches that are alternative to each other. Thereby, *granularity conflicts* are resolved.
3. *Unique elements*: Unique elements belong to alternatives that are defined in only one view and thus are defined in the integrated schema.
4. *Arcs*: There is an arc between two elements in the integrated schema, if and only if there is an arc between the corresponding elements in a view.
5. *Assignment of labels*: A label is assigned to an arc if the corresponding label is assigned to this arc in a view schema.

Example 11. The integrated object life-cycle for the view schemas is shown in Fig. 7. Equivalent elements of the views are represented by one element in the integrated schema RES (e.g., requestBooking, inUse, p). The different kinds of payment are defined as alternative branches. All unique elements are defined in RES. The elements with dotted lines and label s will be explained later.

Each possible life-cycle occurrence in the integrated schema B must be reflected in at least one view. But since B comprises all alternative branches of the views, additional *synchronization states* must be introduced in B to enforce that alternatives can only be executed in the same way as defined in the views.

Example 12. Consider the example shown in Figs. 6 and 7: According to view V_2 , activity sendBill can only be executed if a room reservation has been confirmed before, i.e., activity confirm has been executed. Activities payCash and payByCC can only be executed according to V_3 if activity accept has been executed before. Therefore, a new synchronization label s and synchronization states are introduced to enforce the constraints defined in the views.

These additional labels and synchronization states can be defined in the view schemas B_1 or B_2 during *schema conformation* and integrated in B . A more practical approach will be that — as in the example — synchronization states and labels are introduced after integration if life-cycle occurrences are allowed in B that cannot be observed in any view.

The integrated object life-cycle is a correct integration of the object life-cycles in the views if the modeler has determined correspondences consistently. The check whether B is an observation consistent generalization of B_1 and of B_2 can be automated (cf. [18,19]). Since all elements of the views are integrated in B , the criterion of *no loss of information* is fulfilled. *Correspondence compliance* is fulfilled as all elements that are equivalent to each other are integrated to the same element and all inclusions and subnet correspondences are defined as alternative branches. Further, unique elements are *not* integrated with elements from another view. The *correspondence of life-cycle occurrences* is fulfilled as the set of possible life-cycle occurrences in B is restricted by synchronization states and synchronization labels.

Incorrect object life-cycles may result from contradictory information in the views or inadequate correspondences. Then the modeler must examine once more the set of correspondences, repeat the step of schema conformation, or must even change the view schemas, probably by consulting the modelers of the views.

5 Conclusion

We introduced an integration approach to define the object life-cycle of an object type based on incomplete views of the object life-cycle. View schemas can be incomplete with respect to the considered extensions and intensions. The integration process comprises two phases: First, the extensions considered in view schemas are separated to common and disjoint subclasses and the sub-views are defined for these subclasses. The behavior of common sub-views is defined using integration by specialization. In the second step, sub-views with disjoint extensions are integrated. The main idea of the second step is that the integrated schema includes all alternatives that are defined in any sub-view.

In future research, we will formally define how the integrated schema is generated based on view schemas and a set of correspondences between elements in the view schemas. Formal criteria will be established that must be fulfilled by the set of correspondences such that the integration process yields a correct integrated behavior diagram. Further, tools will be implemented to support the user in integrating view schemas.

References

1. C. Batini, M. Lenzerini, and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986. 414, 422
2. P. Bichler, G. Preuner, and M. Schrefl. Workflow Transparency. In *Proc. 9th Int. Conf. on Advanced Information Systems Engineering (CAiSE '97)*, LNCS 1250. Springer, 1997. 414

3. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. The Addison-Wesley Object Technology Series. Addison-Wesley, 1998. 414
4. O. Bukhres and A. Elmagarmid. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Prentice-Hall, 1996. 414
5. S. Conrad, I. Schmitt, and C. Türker. Considering Integrity Constraints During Federated Database Design. *Proc. 16th British National Conf. on Databases (BNCOD 16)*, LNCS 1405. Springer, 1998. 414
6. L. Ekenberg and P. Johannesson. A Formal Basis for Dynamic Schema Integration. In *Proc. 15th Int. Conf. on Conceptual Modeling (ER '96)*, LNCS 1157. Springer, 1996. 414
7. D. W. Embley. *Object Database Development: Concepts and Principles*. Addison-Wesley, 1998. 414
8. G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A View-Oriented Approach to System Modelling Based on Graph Transformation. In *Proc. 6th European Software Engineering Conf. (ESEC '97)*, LNCS 1301. Springer, 1997. 414
9. H. Frank and J. Eder. Integration of Statecharts. In *Proc. 3rd IFCIS Int. Conf. on Cooperative Information Systems (CoopIS '98)*. IEEE Computer Society, 1998. 414
10. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science Of Computer Programming*, 8:231–274, 1987. 414
11. G. Kappel and M. Schrefl. Object/Behavior Diagrams. In *Proc. 7th Int. Conf. on Data Engineering (ICDE '91)*, 1991. 414
12. W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1(3):251–279, 1993. 421
13. M. Lenzerini. Description Logics and Their Relationships with Databases. In *Proc. 7th Int. Conf. on Database Theory (ICDT '99)*, LNCS 1540. Springer, 1999. 414
14. G. Preuner. *Definition of Behavior in Object-Oriented Databases by View Integration*. PhD Thesis. infix-Verlag, 1999. 414, 422
15. G. Preuner and M. Schrefl. Observation Consistent Integration of Business Processes. In *Proc. 9th Australasian Database Conf. (ADC '98)*, vol. 20 of *Australian Computer Science Communications*. Springer, 1998. 415
16. G. Preuner and M. Schrefl. Observation Consistent Integration of Views of Object Life-Cycles. In *Proc. 16th British National Conf. on Databases (BNCOD 16)*, LNCS 1405. Springer, 1998. 414, 415, 422
17. I. Schmitt and G. Saake. Schema Integration and View Derivation by Resolving Intensional and Extensional Overlappings. In *Proc. 9th ICSA Int. Conf. on Parallel and Distributed Computing Systems*, 1996. 414
18. M. Schrefl and M. Stumptner. Behavior Consistent Extension of Object Life Cycles. In *Proc. 14th Int. Conf. on Object-Oriented and Entity-Relationship Modeling (Oo-ER '95)*, LNCS 1021. Springer, 1995. 414, 415, 417, 428
19. M. Schrefl and M. Stumptner. Behavior Consistent Refinement of Object Life Cycles. In *Proc. 16th Int. Conf. on Conceptual Modeling (ER '97)*, LNCS 1331. Springer, 1997. 414, 415, 416, 418, 428
20. A. Sheth and V. Kashyap. So Far (Schematically) yet So Near (Semantically). In *Proc. DS-5 Semantics of Interoperable Database Systems*, 1992. 421
21. A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990. 414

22. C. Thieme and A. Siebes. Guiding Schema Integration by Behavioural Information. *Information Systems*, 20(4):305–316, 1995. 414
23. M. Vermeer and P. Apers. Behaviour specification in database interoperation. In *Proc. 9th Int. Conf. on Advanced Information Systems Engineering (CAiSE '97)*, LNCS 1250. Springer, 1997. 414

Towards an Automatic Integration of Statecharts

Heinz Frank and Johann Eder

Institut für Informatik-Systeme
Universität Klagenfurt, Universitätsstraße 65-67, A-9020 Klagenfurt
{heinz,eder}@ifi.uni-klu.ac.at
<http://www.ifi.uni-klu.ac.at>

Abstract. The integration of statecharts is part of an integration methodology for object oriented views. Statecharts are the most important language for the representation of the behaviour of objects and are used in many object oriented modeling techniques, e.g. in UML ([23]).

In this paper we focus on the situation where the behaviour of an object type is represented in several statecharts, which have to be integrated into a single statechart. The presented approach allows an automatic integration process but gives the designer possibilities to make own decisions to guide the integration process and to achieve qualitative design goals.

1 Introduction

Conceptual modeling of a universe of discourse using an object oriented data model has two dimensions: the structure of objects and their relationships are represented in a static model (or object model) and the behaviour of objects is documented in a dynamic model ([3,7,24]).

Statecharts, introduced by David Harel ([15,16,17]), are a popular method for designing the behaviour of objects. This concept is used in various design methodologies, e. g. OMT ([24]), OOD ([3]) or UML ([4,23]).

View integration is a commonly followed technique for developing a conceptual model. The universe of discourse is described from the viewpoint of different user groups or parts of the systems resulting in a set of external models. In a second step these models are integrated into a common conceptual schema.

Many integration methodologies consider only the structure of objects, their attributes and the relationships between them. A survey of such approaches can be found in [5]. Some approaches also deal with the integration of methods (e. g. [13], [27]), but little work has been done concerning the integration of behavioural models.

Preuner and Schrefl ([22]) discuss the integration of object life cycles. They use so called *Object/Behaviour Diagrams* to represent the behaviour of objects. In difference to our approach they assume that states and activities can be identified by their names. Common behaviour of objects, described in different views, can be identified by the names of the constructs. Conflicts, such as naming conflicts, must be solved before the integration process. In our approach states

and transitions are identified by using logical conditions (see section 2). Of course we have to deal with naming conflicts, e. g. that different events have the same name. However, such conflicts do not influence the integration process. They can be solved even at the end of the integration process. Furthermore, the usage of logical conditions allows to automate the integration process.

In an earlier paper ([10]) we presented an overview of our whole integration methodology for object oriented views. For this method we assume that models have been developed from different perspectives. Each of these view models consists of a structural (or static) model and a set of behavioural (or dynamic) models in form of statecharts (for each type one). Our integration methodology consists of two major phases, the *integration of the static models* and the *integration of the dynamic models*.

The integration of the static models deals with the structural parts (types, attributes and their relationships to other types). The aim of this phase is to identify and solve conflicts (naming conflicts or structural conflicts) among the types of the various views. The result of this integration phase is the common conceptual static model of the universe of discourse. Several integration strategies, mainly for the entity relationship model ([6]), were published in the past, e. g. [1,13,14,20,26]. Further comparative analysis of view integration methodologies were made in [2] and [25]. For our methodology we have not yet developed another strategy for integrating the structural part of types but use already published methodologies, mainly that one of Navathe et al. ([20,21]).

The integration of the static models results in an integrated conceptual model, a common agreement about types, their internal structure (attributes) and their relationships. Afterwards the integration of the dynamic models takes place. The input parameters are one integrated type and its various statecharts, describing the behaviour of this type from different viewpoints. The aim of this integration phase is to obtain the common behaviour of this type.

For the integration of statecharts we propose two phases:

- *The-integration-in-the-large*: In this integration phase an overall structure of the integrated statechart is developed. All statecharts of the integrated type are analyzed simultaneously in order to compute an integration plan. The integration plan consists of a tree of integration operators. Each operator has two statecharts as input and computes a statechart integrating both. Some of the integration operators can be performed automatically without the aid of a designer. In these cases the statecharts to integrate overlap only on marginal states (these are start and end states). The integration operators simply merge these marginal states to integrate the statecharts. The goal is to develop an integration plan with minimal integration effort, i. e. with minimal interactions with the designer.

The integration-in-the-large phase was subject of an earlier paper ([12]), where we showed that this integration phase can be performed automatically without the aid of a designer.

- *The-integration-in-the-small*: According to the integration plan, the integration operators are carried out step by step. However, in some situations

we need a more detailed analysis of the involved statecharts and probably designer decisions.

The scope of this paper is the *integration-in-the-small*. We concentrate on the situation where we have to deal with one integrated type whose behaviour is described by two different statecharts. The result is the integrated statechart. In this paper however, we omit some formal details and proofs. Interested readers are referred to [9] and [11].

The paper is organized as follows: In section 2 we present an overview of the data model we use and discuss some extensions we made to the statechart language. In section 3 an example from the domain of a library is presented to demonstrate our methodology. The integration process for statecharts is shown in section 4, illustrated by the library example. In section 5 we draw some conclusions.

2 The Data Model

For the structural part of types we are using a very simple data model (according to [8]). A type is a labeled set of type attributes. E. g. `type book = [title: string, pages: integer]` is a type where *book* is name of the type. *Title* and *pages* are the attributes of the type. Attributes are typed with basic types, such as *string* or *integer* or using labels of user defined types.

For the representation of the behaviour of a type we are using the statechart language ([15,16,17]). A statechart of a type primarily consists of *states*, *events* and *transitions*. The major elements of statecharts are shown in figure 1.

A *state* is a condition or situation during the life time of an object during which it satisfies some condition. An object satisfying the condition of a state, is said to be in that state ([23]). We call this condition the *range* of a state.

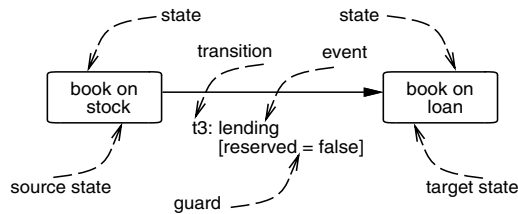


Fig. 1. The basic elements of statecharts

A *transition* is a kind of relationship between two states and is triggered by an *event*. A transition indicates that an object which is in the first state (called *source state*) will enter the second state (called *target state*) when the event occurs and some specified condition (called the *guard* of the transition) holds ([23]). The conjunction of the range of the source state of the transition and

its guard is called the *precondition* of the transition. At the end of a transition the object satisfies the *postcondition* of the transition, which must imply the range of the target state of the transition.

As specification language for these conditions (the range of states, pre- and postconditions and guards of transitions) we use $\mathcal{TQL}++$ ([18,19]). The language allows the definition of logical conditions, which objects have to satisfy. E. g. the range of the state solvent of a type bank in $\mathcal{TQL}++$ would be *this.assets* > 0. To be an object of this type in this state the value of its attribute *assets* must be greater than zero.

As notation $S_1.Range()$ is used for the range of the state S_1 . We use $t.PreC()$ for the precondition of a transition t . As these conditions are logical expressions we may combine them by disjunction, conjunction or negation. For instance, the precondition of a transition t is the conjunction of the range of its source state and its guard, that is $t.Source.State.Range() \wedge t.Guard()$.

Ranges of states as well as pre- and postconditions of transitions are used to define the semantics of statecharts. We developed a complete set of schema transformations to transform a statechart into any other equivalent statechart ([11]). As an example we have transformations to decompose and to construct state hierarchies, to split and to combine states and to shift transitions within state hierarchies. These schema transformations are used in the integration process to prepare the statecharts and to integrate them.

In [11] we have shown that any statechart with state hierarchies can be transformed into an equivalent statechart without state hierarchies. For the integration we assume statecharts without state hierarchies.

Details about the semantics of statecharts and schema transformations can be found in [11] including the proofs that the schema transformations preserve the semantics of statecharts.

3 The Example

Let us introduce a short example from the domain of a library showing the behaviour of books from the viewpoint of two departments. This example is used later to discuss each integration step.

Assume that the integration of the static models results into an integrated type *Book*, having the following syntax:

```
Book = [
  Title: str,
  Authors: {Author},
  reserved: bool,
  status: (new, in library, borrowed, in textbook collection)]
```

The behaviour of a book from the viewpoint of the Service Desk Department is shown in figure 2(a). The department registers new books and places them into the library. Books can be borrowed if they are available. Borrowed books are returned into the library.

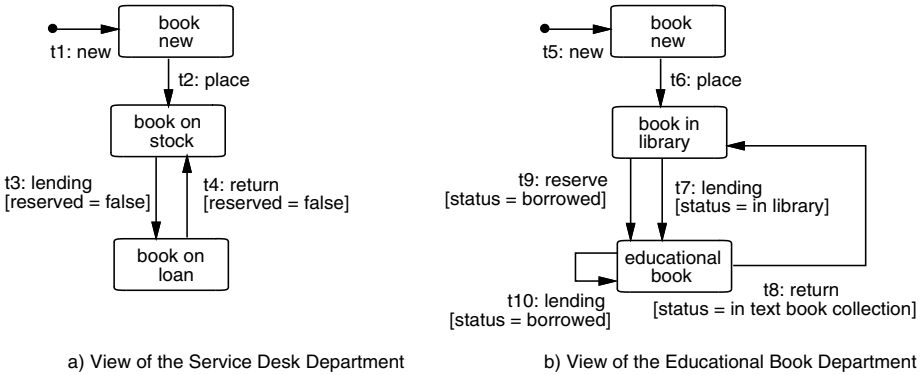


Fig. 2. The behaviour of the type book

Table 1. Ranges of the states of the Service Desk Department

book new	this.status = new
book on stock	this.status = in library $\wedge$ this.reserved = false
book on loan	this.status = borrowed

The second department is the Educational Book Department which registers new books and places them into the library too. Furthermore, it is responsible for the administration of educational books which are necessary for a lecture during a certain period. These books may not be borrowed by anyone until the end of the lecture. If a book in question is out of stock it can be reserved by the department. Such books may not be borrowed by anyone except the educational book department. If a book, borrowed by the department, is no longer necessary for a lecture it is returned. The behaviour of a book from the viewpoint of this department is shown in figure 2(b).

For the integration we require the specifications of the ranges of states which are shown in the tables 1 and 2. The postconditions of the transitions are either equivalent to the range of their target state or shown in table 3. Note that for the specifications of the guards we omit the keyword *this* in the figures.

4 The Process of Integrating Statecharts

4.1 Overview

According to the integration plan, which was developed in the integration-in-the-large phase, we have to deal with one integrated type and two statecharts to integrate. Structural conflicts, such as naming conflicts, have already been solved in previous integration steps. Therefore, we do not have to consider naming

Table 2. Ranges of the states of the Educational Book Department

book new	this.status = new
book in library	(this.status = in library $\vee$ this.status = borrowed) $\wedge$ this.reserved = false
educational book	(this.status = borrowed $\wedge$ this.reserved = true) $\vee$ this.status = in text book collection

Table 3. The postconditions of the transitions

t3:lending	this.status = borrowed $\wedge$ this.reserved = false
t6:place	this.status = in library $\wedge$ this.reserved = false
t7:lending	this.status = in textbook collection
t8:return	this.status = in library $\wedge$ this.reserved = false
t9:reserve	this.status = borrowed $\wedge$ this.reserved = true
t10:lending	this.status = in textbook collection

conflicts between attributes used in the definitions of the ranges of states, pre- and postconditions or guards of transitions.

The integration process resulting in the integrated statechart consists of three major phases (shown in figure 3):

- The *schema analysis* phase: In this phase the statecharts are analyzed to find common behaviour which is represented in a graph called the *state relationship graph*.
- The *schema transformation* phase: The statecharts are transformed so that common behaviour is represented by a single state in both statecharts.
- The *schema merging* phase: The transformed statecharts are merged to the integrated statechart.
- The *schema restructuring* phase: The integrated statechart is restructured to meet quality criteria and to make it more readable. The result of this step is the final statechart.

4.2 The Schema Analysis Phase

In the schema analysis phase the schemas are analyzed to find behaviour which is described in both schemas. The ranges of the states are used to detect common behaviour. We define four classes of relationships between states:

- Two states are *disjoint* if their ranges exclude each other. No object can satisfy the ranges of both states simultaneously.
- Two states are *equivalent* if their ranges are equivalent. Each object either satisfies both ranges or neither.

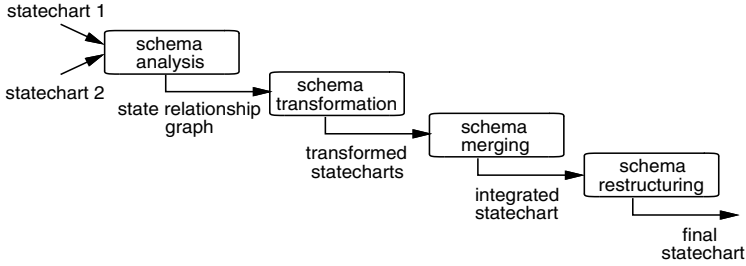


Fig. 3. The process of integrating two statecharts

- A state S_1 *subsumes* a state S_2 if the range of S_2 implies the range of S_1 . An object, satisfying the range of S_2 , satisfies the range of S_1 too.
- Two states are *overlapping* if their ranges overlap. If an object satisfies the range of one of the states, it might but need not satisfy the range of the other state.

Example: The states **book new** from the **book ordering department** and **book new** from the **book book service desk** are equivalent as their ranges are equivalent ($this.status = new$). The range of state **book on stock** ($this.status = in\ library \wedge this.reserved = false$) implies the range of **book in the library** ($(this.status = in\ library \vee this.status = borrowed) \wedge this.reserved = false$). The states **book on loan** ($this.status = borrowed$) and the state **educational book** ($(this.status = borrowed \wedge this.reserved = true) \vee this.status = in\ text\ book\ collection$) overlap. (compare table 1 and 2). $\square$

The relationships between states are used to build the *state relationship graph*. Nodes of the graph correspond to states of the statecharts, edges represent the relationship between two states and are annotated with the kind of relationship. An edge between two nodes exists if the corresponding states are not disjoint. Note that states of the same statechart must be disjoint if the statechart is correct ([17]). Figure 4 shows the state relationship graph of our library example.

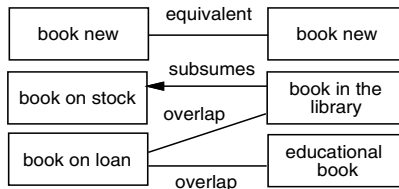


Fig. 4. The state relationship graph

4.3 The Schema Transformation Phase

The goal of the schema transformation phase is to transform the statecharts such that common behaviour is described by a single state in both statecharts, i. e. until the state relationship graph contains only 'equivalent' edges between nodes.

For transforming the statecharts we are using the schema transformation $Split(S, S_1, S_2, B_1, B_2)$ which replaces a state S with two states: S_1 with the ranges B_1 and S_2 with the range B_2 , where $B_1 \vee B_2$ is equivalent to the range of S .

A transition t , having S as source state, is replaced with two transitions t_1 with source state S_1 and t_2 with source state S_2 . Transitions, having S as target state, are replaced with two transitions in an analogous way. However, in order to preserve the semantics, the postcondition of t_1 is replaced with the conjunction of the postcondition of t and the range of its new target state S_1 . Similar, the postcondition of t_2 is replaced with the conjunction of the postcondition of t and the range of S_2 . Note that transitions having S as source and target state are replaced with four transitions.

After the transformation some of the copied transitions might have contradictory pre- or postconditions, e. g. if it is not possible for any object to satisfy the range of a source state and the guard of a copied transition simultaneously. Such transitions can be deleted, without changing the semantics of the statechart. More formal details about the schema transformation including the proof that $Split$ preserves the semantics of a statechart can be found in [11].

This schema transformation and the information represented in the state relationship graph are used to transform the statecharts. We have to consider two cases: a state is subsumed by another state and states which overlap.

Transformation of subsuming states. A state S , subsuming a state T of the other statechart, is transformed into two states S_1 and S_2 such that the ranges of S_1 and T are equivalent and the ranges of S_2 and T are disjoint. We need the transformation

$$1. \text{Split}(S, S_1, S_2, T.Range(), S.Range() \wedge \neg T.Range())$$

$Split$ demands that $S.Range() \equiv T.Range() \vee (S.Range() \wedge \neg T.Range())$. This is obvious, as the range of T implies the range of S .

In the state relationship graph the edge between S and T is deleted. The node S is replaced with the nodes S_1 and S_2 . Obviously, S_1 has an equivalent relationship to T , but is disjoint to all other states. The node S_2 may have relationships to other states, if S had further (other than disjoint) relationships.

Example: According to the state relationship graph (figure 4) the state `book` in the library subsumes the state `book on stock`. The state is split into two states with $Split(book\ in\ the\ library, book\ in\ the\ library(1), book\ in\ the\ library(2), B_1, B_2)$. B_1 is the range of the state `book on stock`: `this.status = in library` $\wedge$ `this.reserved = false`. B_2 becomes `book in the library.Range()  $\wedge$   $\neg$  book on stock.Range()` which

is equivalent to $this.status = borrowed \wedge this.reserved = false$. It is easy to see that the disjunction of B_1 and B_2 is equivalent to the range of the state **book** in the library (see tables 1 and 2). The ranges of the new states:

book in the library (1)	$this.status = in\ library \wedge this.reserved = false$
book in the library (2)	$this.status = borrowed \wedge this.reserved = false$

The result of the transformation is shown in figure 5. Note that all transitions starting from or ending in the original state are duplicated, their source and target states are adopted. To make it more readable we numbered them: $t6(1):place$ is the first copy of the transition $t6:place$, $t6(2):place$ the second one.

Lets take a closer look to the guard of the transition $t9(1):reserve$, which is $this.status = borrowed$. However, the range of the source state of this transition is $this.status = in\ library \wedge this.reserved = false$. As the precondition of a transition is the conjunction of the range of its source state and its guard, no object can satisfy the precondition of the transition $t9(1):reserve$. We delete this transition and in analogy the transition $t7(2):lending$.

We have to consider the changed postconditions of the new transitions, e. g. the transition $t8(2):return$. The transformation replaces the postcondition of transitions, having the split state as target state, with the conjunction of its original postcondition and the range of its new target state. The postcondition of the transition $t8:return$ is $this.status = in\ library \wedge this.reserved = false$ (see table 3). Taking the range of the state **book in the library(2)** we see that no object is able to satisfy the conjunction of the postcondition of $t8:return$ and the range of this state. The transition $t8(2):return$ is deleted and in analogy the transition $t6(2):place$.

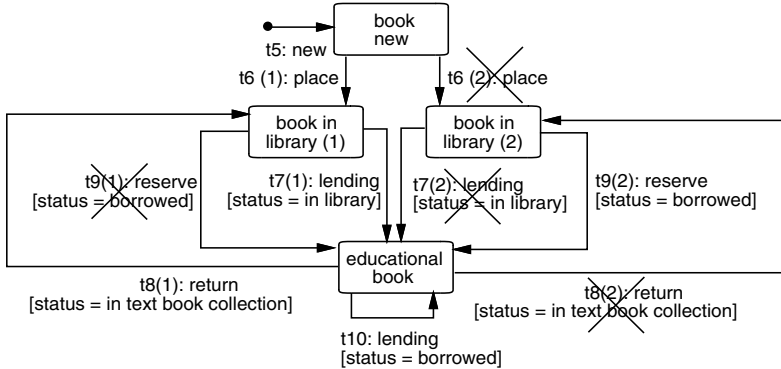


Fig. 5. Transforming the view of the Educational Book Department using the transformation *Split*

Finally, we have to change the state relationship graph shown in figure 6. Note that the relationship between **book on loan** and **book in the library (2)** is now 'subsuming' (compare table 1). $\square$

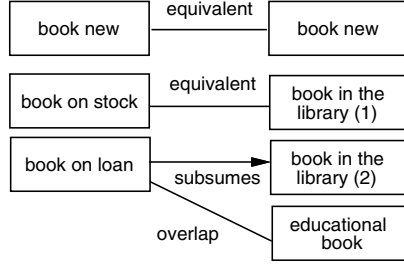


Fig. 6. The state relationship graph after transforming the view of the Educational Book Department

Transformation of overlapping states. In the case that two states S and T overlap, we split them into four states. Two states are disjoint to each other and two states are equivalent. The transformations:

1. $Split(S, S_1, S_2, S.Range() \wedge \neg T.Range(), S.Range() \wedge T.Range())$
2. $Split(T, T_1, T_2, T.Range() \wedge \neg S.Range(), T.Range() \wedge S.Range())$

The schema transformation *Split* demands that $S.Range() \equiv (S.Range() \wedge \neg T.Range()) \vee (S.Range() \wedge T.Range())$ which is obvious. It is easy to see that the analogous condition holds for the second transformation too.

In the state relationship graph S and T are replaced with nodes which correspond to the new states. Note that the states S_2 and T_2 are equivalent but disjoint to all other states. The states S_1 and T_1 might have relationships to other states (if S or T had further relationships other than disjoint ones to other states).

Example: In the state relationship graph shown in figure 6 we have an 'overlapping' relationship between the states *book on loan* and *educational book*. The states are split as shown in figure 7 and figure 8.

The state *book on loan* is split using the transformation $Split(book\ on\ loan, book\ on\ loan(1), book\ on\ loan(2), B_1, B_2)$ where $B_1 = book\ on\ loan.Range() \wedge \neg educational\ book.Range()$ and $B_2 = book\ on\ loan.Range() \wedge educational\ book.Range()$. The new states with their ranges:

book on loan (1)	this.status = borrowed $\wedge$ this.reserved = false
book on loan (2)	this.status = borrowed $\wedge$ this.reserved = true

The transition *t4(2):return* is deleted (figure 7) because its guard does not match with the range of its source state. The transition *t3(2):lending* is deleted because its postcondition does not imply the range of its source state (compare table 3).

educational book (1)	this.status = in textbook collection
educational book (2)	this.status = borrowed $\wedge$ this.reserved = true

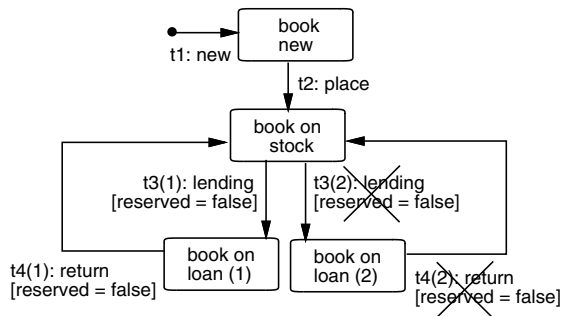


Fig. 7. Transforming the view of the Service Desk Department using the transformation *Split*

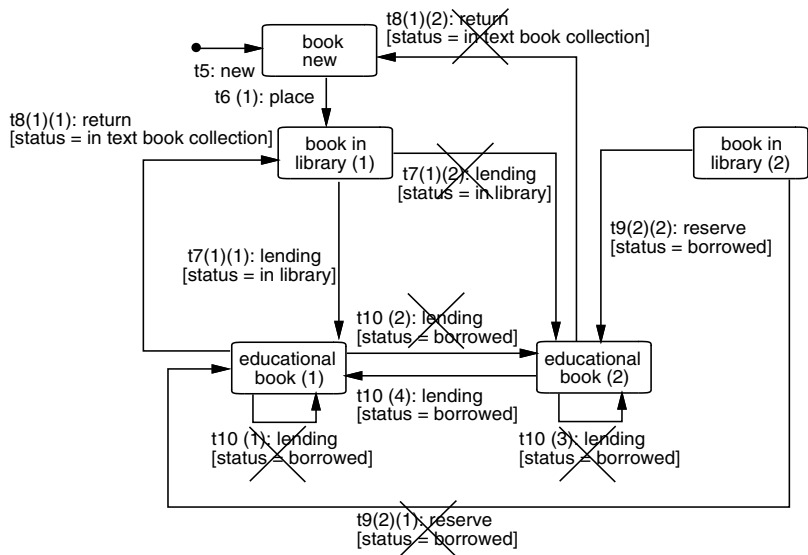


Fig. 8. The second transformation of the view of the Educational Book Department using the transformation *Split*

The state **educational book** is split using the transformation $Split(educational\ book, educational\ book(1), educational\ book(2), B_1, B_2)$. The parameter B_1 is $educational\ book.Range() \wedge \neg book\ on\ loan.Range()$. B_2 is $educational\ book.Range() \wedge book\ on\ loan.Range()$. The new states with their ranges:

Note that the transition **t10:lending** is replaced with four transitions as the state **educational book** is source and target state of this transition (see figure 8).

The transitions **t10(1):lending** and **t10(2):lending** are deleted because the range of their source states and their guards contradict. The postcondition of the transition **t9(2)(1):reserve** (see table 3) does not imply the range of its target state and is removed too.

The transition **t7(1)(2):return** has a guard which does not match with the range of its target states. The transitions **t10(3):lending** and **t7(1)(2):lending** have postconditions which do not imply the range of their target states (compare table 3). All these transitions are deleted.

Finally, we change the state relationship graph again (figure 9). Note that the states **book on loan(1)** and **book in the library(2)** now have equivalent ranges ($this.status = borrowed \wedge this.reserved = false$). The schema transformation phase of our example is completed, the state relationship graph contains only nodes representing disjoint or equivalent states. $\square$

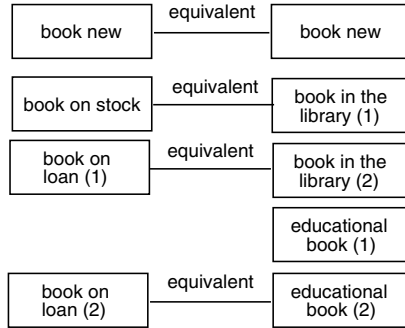


Fig. 9. The final state relationship graph

With the presented schema transformations the statecharts are transformed as long as the state relationship graph contains subsuming or overlapping edges between nodes. In each step common behaviour is extracted and a subsuming or overlapping edge of the relationship graph is removed. The transforming process terminates in any case.

4.4 The Schema Merging Phase

In this phase the transformed statecharts are merged into the integrated statechart. According to the state relationship graph, states are either disjoint or

equivalent. The merging of the statecharts is quite easy. Two equivalent states are merged to a single state, disjoint states remain unchanged.

Example: Let us complete the integration process of our library example. The statecharts (figure 7 and figure 8) are merged. Equivalent states such as book on stock and book in the library(1) (figure 9) are merged to single states. The integrated statechart is shown in figure 10. $\square$

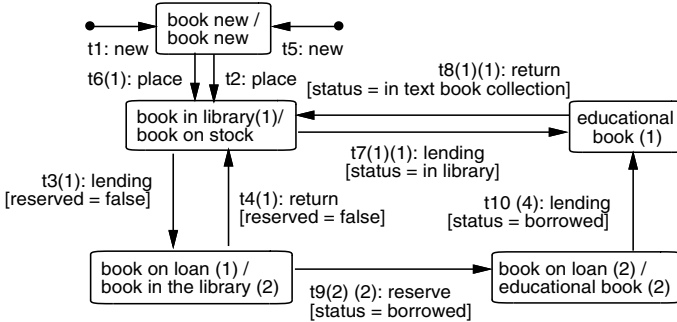


Fig. 10. The integrated statechart

During the integration process we used schema transformations that transform states into disjoint states. As shown in [11] a state of a statechart can always be split into two disjoint states without changing the semantics of the statechart. It is easy to see that combining equivalent states does not change the information described by a statechart. The information, described by the statecharts which must be integrated, is described by the integrated statechart too.

4.5 The Schema Restructuring Phase

The schema merging step delivers a first cut integrated statechart. Since we made a series of transformations to extract common behaviour, the readability of the integrated statechart should be improved, e g. redundant information should be removed.

For instance, the transitions `t1:new` and `t5:new` have equivalent pre- and postconditions and the same target states. Assuming that they are triggered by the same event, i. e. `new`, we may combine them to one transition. Otherwise the designer should rename one event. Analogous the transitions `t2:place` and `t6(1):place` can be combined to a single transition.

Before starting with the integration we have decomposed state hierarchies. In the schema restructuring phase state hierarchies can be introduced again. For instance, if states have some transitions triggered by the same event with the same source (or target) state in common, they could be put into a state generalization.

We can restructure the statechart automatically using some quantitative design goals like minimizing the number of edges, or minimizing the number of necessary guards. The final restructuring and naming, however, should be made by a (human) designer to meet qualitative design goals like readability.

5 Conclusion

In this paper we presented an approach for integrating statecharts, which is part of an integration methodology for object oriented views. We assume that the integration of the structural parts of an object oriented data model (i. e. the integration of the types) has already be done.

We concentrate on the situation where the behaviour of one type is described by two different statecharts. We show, how common behaviour can be found and how the different statecharts are merged into an integrated statechart. The integration process follows a very simple strategy: whenever two states are not disjoint, they are transformed into disjoint states and the common ground of these states is put into a separate state.

It was our aim to relieve the designer by automating the integration as much as possible. The major steps of our integration methodology can be done without the aid of a designer. The schema analysis phase, the schema transformation phase, and the schema merging phase could be performed by a tool. The crucial part of such a tool is the language, which is used to specify the range of states and the pre- and postconditions and guards of transitions. Assuming that this language is decidable, the integrated statechart can be computed. However, the designer is needed to make changes in order to improve the quality in the schema restructuring phase, e. g. to make it more readable.

At the moment we are working on a language for representing conditions on states and transitions which is decidable. In a next step we are planning to implement our integration methodology.

We see the main advantages of our approach in the formal treatment of the integration process which allows an automatic integration while giving the designer the possibilities to make decisions and their consequences are carried out automatically in the model.

References

1. Batini, C., Lenzerini, M.: A Methodology for Data Schema Integration in the Entity Relationship Model. *IEEE Transactions on Software Engineering*, 10(6):650–664, November 1984. 431
2. Batini, C., Lenzerini, M., Navathe, S.: A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986. 431
3. Booch, G.: *Object-Oriented Design with Applications*. Benjamin Cummings, 1991. 430, 430
4. Booch, G., Rumbaugh, J., Jackobson, I.: *The Unified Modeling Language: User Guide*. Addison-Wesley, 1999. 430

5. Bukhres, O., Elmagarmid, A. (eds.): Object-Oriented Multidatabase Systems: A Solution for Advanced Applications. Prentice Hall, 1996. 430
6. Chen, P.: The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transaction on Database Systems*, 9–36, March 1976. 431
7. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jere-maes, P.: Object-Oriented Development: The Fusion Method. Prentice Hall Object-Oriented Series. Prentice-Hall, Inc, 1994. 430
8. Formica, A., Groger, H., Missikoff, M.: Object-Oriented Database Schema Analysis and Inheritance Processing: A Graph-Theoretic Approach. *Data- and Knowledge Engineering*, 24:157–181, 1997. 432
9. Frank, H.: View Integration für objektorientierte Datenbanken. Dissertationen zu Datenbanken und Informationssystemen Band 32. infix, 1997. 432
10. Frank, H., Eder, J.: Integration of Behaviour Models. *Proceedings of ER'97 Workshop on Behavioral Modeling and Design Transformations*. <http://osm7.cs.byu.edu/ER97/workshop4>, November 1997. 431
11. Frank, H., Eder, J.: Equivalence of Statecharts. Technical report, Institut für Informatik-Systeme, Universität Klagenfurt, 1998. 432, 433, 433, 433, 437, 442
12. Frank, H., Eder, J.: Integration of Statecharts. In Halper, M. (ed.): *Third IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*. 364–372. IEEE Computer Society, August 1998. 431
13. Geller, J., Perl, Y., Neuhold, E., Sheth, A.: Structural Schema Integration with Full and Partial Correspondence Using the Dual Model. *Information Systems*, 17(6):443–464, 1992. 430, 431
14. Gotthard, W., Lockemann, P., Neufeld, A.: System Guided View Integration for Object-Oriented Databases. *IEEE Transaction on Knowledge and Data Engineering*, 4(1):1–22, January 1992. 431
15. Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987. 430, 432
16. Harel, D.: On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May 1988. 430, 432
17. Harel, D., Naamad, A.: The Statemate Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996. 430, 432, 436
18. Lam, H., Missikoff, M.: On Semantic Verification of Object-Oriented Database Schemas. *Proceedings of Int. Workshop on New Generation Information Technology and Systems - NGITS*, 22–29, June 1993. 433
19. Missikoff, M., Toaiti, M.: Mosaico: An Environment for Specification and Rapid Prototyping of Object-Oriented Database Applications. *EDBT Summer School on Object-Oriented Database Applications*, September 1993. 433
20. Navathe, S., Elmasri, R., Larson, J.: Integrating User Views in Database Design. *IEEE Computers*, 185–197, January 1986. 431, 431
21. Navathe, S., Pernul, G.: *Advances in Computers*. Vol. 35, Chapter Conceptual and Logical Design of Relational Databases, 1–80. Academic Press, 1992. 431
22. Preuner, G., Schrefl, M.: Observation Consistent Integration of Views of Object Life-Cycles. In Embury, S., Fiddian, N., Gray, W., Jones, A. (eds.): *16th British National Conference on Databases. Lecture Notes in Computer Science*, Vol. 1405, 1998. 430
23. Rational Software et.al.: Unified Modeling Language (UML) Version 1.1. <http://www.rational.com/uml>, September 1997. 430, 430, 432, 432
24. Rumbaugh, J., Blaha, M., Premierlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modeling and Design*. Prentice Hall International, Inc, 1991. 430, 430

25. Schrefl, M.: A Comparative Analysis of View Integration Methodologies. In Traunmüller, R., Wagner, R., Mayr, H. (eds.): Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen. 119–136, 1987. Fachtagung EMISA. 431
26. Sheth, A.: Issues in Schema Integration: Perspective of an Industrial Researcher. ARO-Workshop on Heterogeneous Databases, September 1991. 431
27. Thieme, C., Siebes, A. Guiding Schema Integration by Behavioural Information. Information Systems, 20(4):305–316, 1995. 430

Design Support for Database Federations^{*}

Kerstin Schwarz, Ingo Schmitt, Can Türker, Michael Höding, Eyk Hildebrandt,
Sören Balko, Stefan Conrad, and Gunter Saake

Institut für Technische und Betriebliche Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D-39016 Magdeburg, Germany
`sigmafdb@iti.cs.uni-magdeburg.de`

Abstract. Federated database systems provide a homogeneous interface to possibly heterogeneous local database systems. This homogeneous interface consists of a global schema which is the result of a logical integration of the schemata of the corresponding local database systems and file systems. In this paper, we sketch the integration process and a set of tools for supporting the design process. Besides the classical database schema integration, the design process for federated information systems requires the integration of other aspects like integrity rules, authorization policies and transactional processes. This paper reports on an integrated approach to tool support of several of these integration aspects. The different integration facets are linked via the database integration method GIM allowing a high degree of automatic integration steps.

1 Introduction

The **SIGMA**_{FDB} project [26] is the focus of our research in the field of federated database systems. Our common view to federated database systems is a tightly coupled approach based on the five-level schema architecture of Sheth and Larson [24]. This is motivated by the requirement of data consistency which can only be appropriately guaranteed in a tightly coupled environment. A federated database should correctly reflect the modeled real-world. Inconsistencies between the databases concerning the same real-world entities have to be removed. Correct conflict resolution builds the basis for a correct database integration.

Our research interests as well as the related implementation activities are stimulated by the following application scenarios:

- The first scenario comes from the area of factory planning. Here, different specific application programs (tools) are used for optimizing machine configuration and transport facilities in factories. These applications have been developed independently. In order to enable interoperation between these tools (in particular to support concurrent engineering) an integrated data layer is needed. A main challenge is that most tools store their data in files,

^{*} This research was partially supported by the German State Sachsen-Anhalt under FKZ: 1987A/0025 and 1987/2527R.

each using its own way of structuring the data. Therefore, the files have to be analyzed for building a description of the file structure.

- Our second integration scenario is motivated by the bio-informatics research area where we are faced with highly heterogeneous databases which contain bio-molecular data representing results of thirty years of experimental research. The integration of these partly overlapping databases promises a new quality of databases and enables new applications to access heterogeneous data in a uniform and transparent way [14].
- Our third (and most recent) integration scenario is given by the *Global-Info* project which aims at supporting digital libraries by federation services. Here, the existing systems mainly have WWW interfaces which usually restrict the access to pieces of information in a certain way.

This paper surveys our approach to designing federated database schemata. We briefly describe the tool-set **SIGMA**_{Bench} which supports the design process. The main part of this process obviously comprises the schema integration. For integrating schemata of different local systems these schemata must be available. In our application scenarios we are faced with files which do not offer a schema. Hence, the file structure must be analyzed in order to exploit a schema. The integration is performed following the GIM-Method (Generic Integration Model) [20]. For a correct integration, we have to consider both intensional and extensional conflicts in a common framework. Extensional assertions are specified by the database designer. This is a hard task for the designer and demands a detailed knowledge about the local systems. Since local integrity constraints reduce the set of possible database states, these constraints can be applied to give hints for extensional assertions between local classes [27].

A further step of the design process considers local integrity constraints as additional semantic information for schema integration. In addition, local access rights must also be considered in order to derive global access rights. Last but not least, we regard the design of global transactions and their decomposition into subtransactions as an important task in building federated information systems.

The remainder of this paper is organized as follows. In Section 2 we give an overview of the design process and current state of **SIGMA**_{Bench}. Afterwards, the file analyzer and the derivation of file schemata is described. The automatic generation of extensional assertions is subject of Section 4. Section 5 discusses schema integration including integrity constraints and access rights whereas Section 6 addresses basic issues of designing global transactions.

2 Federation Design Process

Federated database systems [24] provide a homogeneous interface to possibly heterogeneous local database systems. This interface consists of a global schema which is the result of a logical integration of the involved local schemata [3]. The integration is performed in several steps being sketched in this section including file structure analysis, treatment of integrity constraints and access rights.

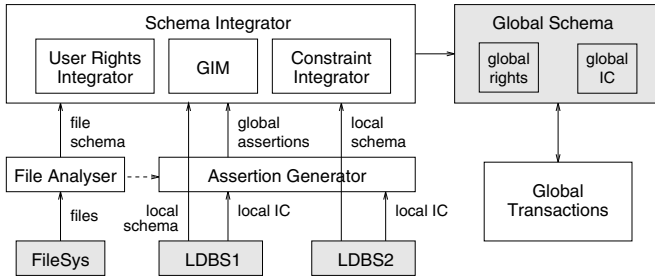


Fig. 1. Components of the Federated Schema Design Tool

Fig. 1 sketches the steps of developing an integrated schema. On the first level, there are the local systems including file clusters and database systems. The assertion generator takes the local integrity constraints for the local (file) schemata and generates a set of extensional assertions. Obviously, these assertions can only be complete if the local integrity constraints are fully specified. In practice, the constraints defined on a schema will probably not be sufficient to express all extensional relationships. However, they may serve as an indication for the federated database designer. The next step comprises the schema integration. Since integrity constraints and access rights influence the global schema, database integration must also comprise the integration of integrity constraints and access rights. Before integrating schemata the integrator has to complete (and probably correct) the generated global assertions. The GIM-Method deals with n-ary extensional assertions. In contrast, most other schema integration approaches are based on only binary extensional assertions which cannot express all possible situations. The result is a global schema with global access rights and integrity constraints. Each access onto the local systems over the global interface has to be done with respect to the global schema. Thus, global transactions may have different dependencies among them determined by the global assertions.

With respect to this process we implemented the federated database design tool **SIGMA_{Bench}** (see Fig. 2). This tool was implemented according to a client-server architecture. Central inter-operation platform is the repository that is realized using a relational DBMS (*YARD*). Clients for dedicated tasks are developed in Java. The *file structure analyzer* comprises different tools for the file structure analysis. Key words, tokens, and brackets can be found semi-automatically based on text analysis. In this way, the design of grammars describing the structure of file clusters is supported. The *schema loader* imports schemata from an Oracle database to our repository. It uses the catalogue tables of the Oracle system and currently considers classes, attributes, data types, and integrity constraints. Current implementation work extends this also to load user profiles including access rights. The *schema editor and viewer* allows the definition and graphical representation of object-oriented schemata for storing them in the repository. Of course, in “real” integration scenarios the schemata should

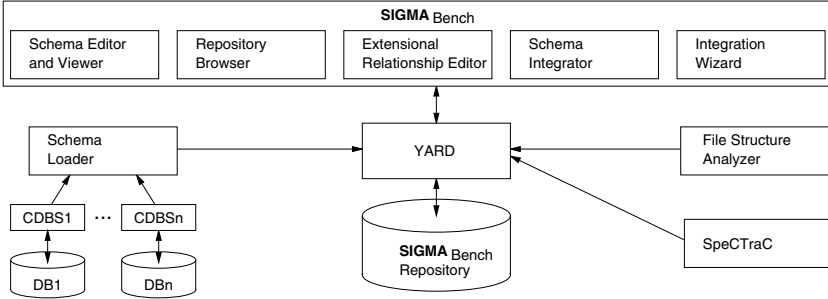


Fig. 2. Architecture of the **SIGMA_{Bench}**

be imported from the local databases. The main issue of the schema editor is to make schemata available for demonstration purposes. Thus, the editor offers classes, attributes, relationships, specialization, and integrity constraints. Currently three kinds of integrity constraints are supported: *uniqueness constraints* like **unique**(Book.Author, Book.Title), *attribute-constant-comparisons* (including **not null**) such as Book.Price > 0.0, and *comparisons between two attributes*, for instance, Book.Title <> Journal.Title. These constraints can be composed to complex integrity constraints by means of logical operators. The *extensional relationship editor* supports the derivation of extensional relationships from integrity constraints (e.g. Journal.ISBN <"1-2345" and Book.ISBN >"2-6789") and extensional assertions (e.g. Library.Book **disjoint to** Project.Book). The extensional relationships are expressed by the GIM-matrix which correlates disjoint extensions and classes of the participating schemata (see Section 5.1). Besides the automatic generation of the matrix using integrity constraints and extensional assertions, our tool also allows its direct manipulation.

The main part of **SIGMA_{Bench}** is the *schema integrator* which automatically derives integrated object-oriented schemata using the source schemata and the modeled extensional relationships. This tool covers extensional and intensional conflicts and supports the integration of integrity constraints. The integration of access rights is currently in development. The *integration wizard* combines all necessary steps for the federated schema generation. It includes the *assertion generator* which currently supports attribute-constant-comparisons and specialization relationships. **SPECTRAC**¹ allows to specify (global) transactions with different dependencies among them. The tool comprises a consistency checker which supports specifying consistent dependencies. In case of an inconsistency, alternative dependencies are proposed. Due to the fixed extensional assertions of the global schema, different commit rules can be derived for global transactions.

¹ A Tool for **S**pecifying **C**onsistent **T**ransaction **C**losures

3 File Structure Analyzing

Traditional federated schema design approaches fail in integrating data files since files neither offer a data model nor a database schema which can be exploited for schema integration. It is important to mention that the integration has to cover all possible files of a given application scenario, and should not be restricted to only one or some example files [13]. In this context, we use the term *file cluster* to refer to these file databases which cover all files of an application system. For the integration of such file clusters we suggest the derivation of a *structure description*, e.g. a grammar, describing the physical structure and the syntax of the data files and a *conceptual schema*, modeling structure, and semantics in database terms. The structure description could be a SGML-DTD (SGML Document Type Definition) or a context-free grammar according to YACC.

Completely structured files can be modeled with a grammar. Several languages for grammar definition exist, supported by tools like lexical scanners and parser generators. Such grammars can directly be compiled into adapters. A promising task is the semiautomated derivation of the grammar and schema. A top-down approach using grammatical inference methods is implemented in the NoDoSe-Tool [1]. Here, the developer defines the grammar by marking data in a text window and assigning them with data structures defined in a grammar or schema window. The tool calculates all candidates for tokens, identifying the data. In that way a structure description can be “mined” step-by-step.

Our approach as a part of the **SIGMA**_{Bench} is a bottom-up approach. The first step is the syntactical analysis of a given set of files. The idea is to separate the words² in a file by delimiters. The set of delimiters can be defined explicitly. The next step is counting the occurrences for each word and sorting the list of words according to the count. Based on the hypothesis, that the most frequent words are good candidates for tokens or key words, this list can support the developer in grammar design. However, the experiments prove, that also non key words can occur very frequently and key words, e.g. identifying classes with very few instances in the files, can occur less frequently. Therefore, the developer has to check and to improve the list manually.

The next syntactical analysis step finds bracket pairs³ or block delimiters. Based on a set of bracket pairs, the *SIGMA-File-Analyzer* constructs a parse tree for the files. The parse tree is translated into a grammar tree by a bottom-up approach beginning with the leaves of the tree. The goal is to find similar structured blocks in all levels of the tree. For each element in a block, two alternative representations are considered. The first one is the *pure* or immediate representation of a word, e.g., **AUTHOR** is represented as **AUTHOR**. This will mainly be used for key words or tokens. The second representation is a *meta*-representation by a type or class, e.g., the texts **Gunter**, **Can**, or **Ingo** can commonly be represented by the associated data type **String**. Checking all combinations of both representations should result in a reduced number of equivalent block patterns.

² A word can also be a single character.

³ Brackets can also be “long” words, e.g., **BEGIN** or **END**.

The following example illustrates the simplification of inner block patterns. The parse tree contains the following inner blocks:

```
AUTHOR Gunter BOOK Databases
AUTHOR Can BOOK Objectbases
AUTHOR Ingo BOOK Schemaintegration
AUTHOR Michael BOOK Oracle8
```

Regarding the number of four elements in the blocks, one can derive the complete set of 16 combinations. The first combination (*pure pure pure pure*) is the source combination. Setting the first element to meta does not reduce the number of blocks. The goal is to calculate the combination that has the fewest different blocks with as few as possible switches from *pure* to *meta*. In the example above, the best combination is (*pure meta pure meta*):

```
AUTHOR String BOOK String
```

In this way, it is possible to reduce the number of different inner blocks. These blocks can now be represented by the calculated combinations.

This algorithm can recursively be applied to each level of the parse tree. The result is a grammar tree which is equivalent to the parse tree. The grammar tree can now be translated into a grammar for a adapter generator. The approach is partially implemented and produces good results for strongly structured files using blocks for structuring. Future work will enhance the technique to files where bracket-pairs are tokens with key word semantics or value semantics.

As argued in [18], a grammar is not a suitable approach for semistructured files. For instance, an HTML file, which is generated by a CGI script accessing a relational database, contains a huge volume of unnecessary and unstructured information in most cases. This information could be control sequences for text formatting, e.g., advertising of products, additional services, etc. Constructing a complete grammar for such a kind of file is both unnecessary and expensive. A more suitable approach for this kind of semistructured files is the application of pattern matching techniques, e.g., with regular expressions. This is done in the FLORID prototype [12] as well as in the WebJDBC system [18].

4 Assertion Generation

Integrity constraints are an important part of database schemata. They describe properties of the modeled real world and thus define the semantics of a database schema. Their enforcement ensures the semantic correctness of the corresponding database states. In order to build and maintain semantically correct database federations, locally existing integrity constraints must be reflected in the global schema. Furthermore, we may use local integrity constraints to derive global assertions for corresponding classes [27]. Extensional assertions are defined in order to relate classes of different schemata according to their extensional relationships. We denote the extension of the class C_1 in a database state S (the

set of current instances) as $Ext_{C_1}^S$. Between the local classes the following extensional assertions [25] can be defined by the database integrator:

Disjointness: $C_1 \oslash C_2 :\Leftrightarrow \forall S: Ext_{C_1}^S \cap Ext_{C_2}^S = \emptyset$

Equivalence: $C_1 \equiv C_2 :\Leftrightarrow \forall S: Ext_{C_1}^S = Ext_{C_2}^S$

Containment: $C_1 \supset C_2 :\Leftrightarrow \forall S: Ext_{C_1}^S \supseteq Ext_{C_2}^S$

Overlap: $C_1 \cap C_2 :\Leftrightarrow true$

Two classes are extensionally disjoint if there are no common objects in the extensions of these classes in any database state. Extensionally equivalent classes have same extensions in each database state. One class extensionally contains another class if the extension of the first class is a superset of the extension of the other class in any database state. The default value is overlap. In this case there are no restrictions on the extensions of the two related classes.

Integrity constraints can be exploited to give hints for extensional assertions between local classes [27]. In the following example, we show the derivation of extensional assertions for the local classes `Library.Book` and `Project.Book`. Suppose these classes consist of books of a library and a project, respectively. Furthermore, both of the local classes have an attribute `Publisher` with the following integrity constraints. In the library, there are only books published by Springer, Addison-Wesley, and Prentice-Hall. The books of the project are published by infix and Shaker:

`Library.Book.Publisher` $\in$ {Springer, Addison-Wesley, Prentice-Hall}

`Project.Book.Publisher` $\in$ {infix, Shaker}

Obviously, the extensions of classes `Library.Book` and `Project.Book` are disjoint. Moreover, the derivation of assertions based on linear arithmetic constraints is currently integrated into the tool *integration wizard*.

5 Integration

Integration concerns three parts, schema integration in general, integration of integrity constraints, and integration of access rights.

5.1 Schema Integration

Schema integration [3] is the main step in federated database design. In correspondence to the five-level-schema architecture [24] the schemata of existing component databases must be integrated to the federated schema. A complex task is to overcome schema heterogeneity. This step typically consists of detecting semantic conflicts and then of solving these conflicts by designing new schemata and corresponding schema mappings. After having solved all conflicts, the schemata can be merged into a federated schema.

In our project we developed the new schema integration method *GIM* [20,21]. GIM is an acronym for *Generic Integration Model*, an intermediate data model.

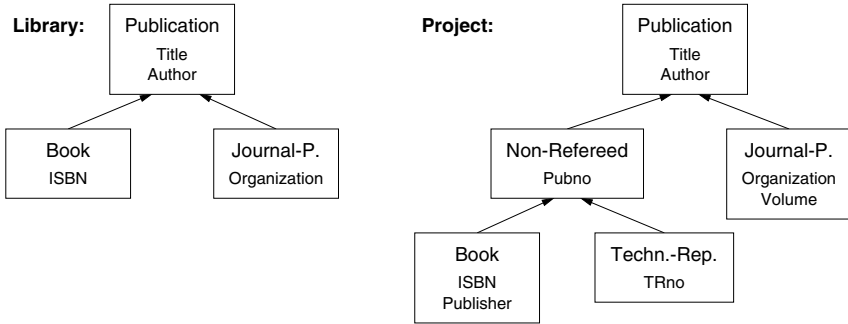


Fig. 3. Library and Project Schema

This model is designed for schema integration. The GIM approach enables the usage of algorithms from formal concept analysis [9] increasing the part of schema integration which can be done automatically.

In the following, the GIM-Method is sketched by an example. Assume, there is a library database storing information about publications, where books and journal papers are special types of publications. Each publication is either a book or a journal paper. This database has to be integrated with a project database that stores project publications. This database distinguishes more types of publications. Fig. 3 depicts the corresponding schemata of both databases.

Before these schemata are integrated, conflicts have to be resolved. In this paper we consider extensional and intensional conflicts. We assume that other types of conflicts are resolved as described, e.g., in [15,25]. Extensional conflicts refer to redundancies among different classes whereas intensional conflicts are caused by different types of semantically related classes. The schemata define extensional subset relationships between sub- and superclasses and relationships due to integrity constraints. In general, it is impossible to completely conclude from given databases and integrity constraints whether the extensions of an arbitrary set of classes from different databases can simultaneously contain objects modeling a real-world object. Therefore, the designer has to give additional information about the extensional relationships among the classes.

Knowing the extensional overlaps the original extensions can be decomposed into disjoint *base extensions*. If, for example, the classes A and B overlap then their extensions are decomposed into three base extensions $(A \setminus B, B \setminus A, A \cap B)$. The algorithm presented in [22] computes the base extensions from the extensional assertions and is implemented in the **SIGMA**_{Bench}.

The base extensions of our example are given in Fig. 4 (left). In our example, the original extensions are partitioned into base extensions 1–9. For example, the extension **Book** of the database **Library** is partitioned into the base extensions 1 and 9. The base extension 9 contains books from the library database which are simultaneously stored in the project database whereas the base extension 1

Local Classes	1	2	3	4	5	6	7	8	9	Attr-Ext	1	2	3	4	5	6	7	8	9
Library.Publication	✓	✓						✓	✓	Title	✓	✓	✓	✓	✓	✓	✓	✓	✓
Library.Book	✓								✓	Author	✓	✓	✓	✓	✓	✓	✓	✓	✓
Library.Journal-Paper		✓						✓		ISBN	✓				✓				✓
Project.Publication			✓	✓	✓	✓	✓	✓	✓	Organization		✓		✓		✓		✓	
Project.Non-Refereed			✓		✓	✓	✓		✓	Volume				✓		✓		✓	
Project.Journal-Paper				✓		✓		✓		Publisher				✓					✓
Project.Book					✓				✓	TRno			✓			✓			
Project.Technical-Report			✓			✓				Pubno			✓		✓	✓	✓		✓

Fig. 4. Extensional Relationships and Extension-Attribute Relation

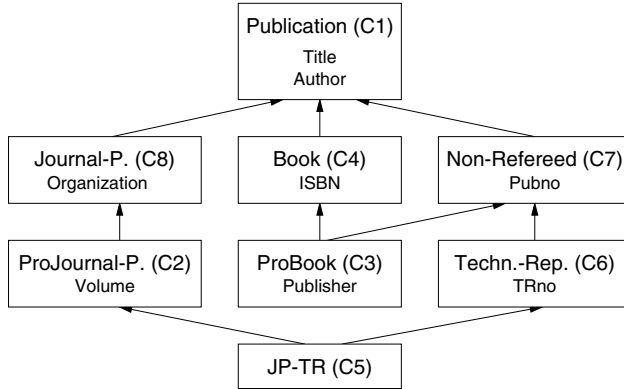
contains the remaining ones. Each publication, which is stored in the library or in the project database, can be assigned to exactly one base extension.

Now we have to compare the intensional aspects of the two schemata. For simplicity, we assume attributes with same names have identical semantics, i.e., *attribute conflicts* [15] are assumed to be resolved. The resolution of attribute conflicts is in general an important step during the integration process, but lies beyond the scope of this paper.

From the given extensional and intensional information we can automatically derive a table that assigns attributes to base extensions. A tick symbol for a column (base extension) and a row (attribute) means that for potential objects of that base extension we know the value of that attribute (Fig. 4 right).

This table representation is a simplified GIM representation of the integrated schemata and can directly be used for further analyses. The order of columns and rows is irrelevant. By exchanging rows and columns we only obtain another representation. In such a representation a rectangle corresponds to a class: a union of base extensions having the same set of attributes. Moreover, we may detect subclass relationships using the rectangle representation of classes. If rectangles overlap, then they are in a specialization relationship. Algorithms of formal concept analysis [9] find all maximal rectangles and extensional subset relations between them. A maximal rectangle represents a class of the federated schema. The algorithm presented in [20], which is a slightly modification of the concept analysis algorithm, finds the following classes (maximal rectangles) from the Fig. 4 (right table):

- C1 = ({1,2,3,4,5,6,7,8,9},{Title,Author})
- C2 = ({4,6,8},{Title,Author,Organization,Volume})
- C3 = ({5,9},{Title,Author,ISBN,Publisher,Pubno})
- C4 = ({1,5,9},{Title,Author,ISBN})
- C5 = ({6},{Title,Author,Organization,Volume,TRno,Pubno})
- C6 = ({3,6},{Title,Author,TRno,Pubno})
- C7 = ({3,5,6,7,9},{Title,Author,Pubno})
- C8 = ({2,4,6,8},{Title,Author,Organization})

**Fig. 5.** Integrated Schema

Now we build the matrix M which represents the irreflexive binary relation $<$. It expresses the specialization relation (subset relation) by comparing the extensions of each pair of classes. Two classes are in specialization relation if their sets of base extensions are in a subset relation. The computation $M_N = M - M \times M$ removes transitive specializations. Only the value ‘1’ in a field of M_N represents a direct specialization relation between two classes.

The classes $C1, \dots, C8$ and M_N can directly be used to produce the federated schema as depicted in Fig. 5. The names of the integrated classes correspond as far as possible to the classes of the two existing databases.

Besides the presented approach there are many other approaches to schema integration, e.g. [7,17,8]. The main difference between our approach and the other ones is the decomposition of extensions into base extensions and the application of efficient algorithms. We are able to integrate different inheritance hierarchies encompassing many classes. The other approaches explain how to integrate very few classes whereas our approach considers the schemata entirely.

5.2 Integrity Constraint Integrator

In order to build and maintain semantically correct database federations, locally existing integrity constraints must be reflected in the global schema. In this context, the notions of *global* and *local understandability* [28] play a central role. Global/local understandability means that a global/local transaction should not be rejected if it fulfills all integrity constraints of the corresponding schema. Our approach to dealing with integrity constraints during schema integration is based on a set of logical rules [6,2]. Integrity constraints are usually given for a certain set of objects, i.e., they correspond to class extensions. Therefore, we always consider constraints together with the extensions for which they are valid. During integration we perform the following steps:

- Based on the GIM-Method, first the base extensions are derived from the given classes. Then, the respective integrity constraints are assigned to the base extensions. In doing so, we have to check which constraints of the original classes are still valid in the corresponding base extensions. Base extensions were created by extensional decomposition. Therefore, we call an integrity constraint *decomposable* if it remains valid on any subset of the original class extension [2]. Any constraint which is decomposable can be adopted unchanged to the corresponding base extensions. Please note that the decomposition of class constraints leads to a loss of information (which have to be compensated later during the composition phase). Obviously, object constraints are also decomposable, e.g., the constraint ($\text{price} < 200.00$) must hold also for the corresponding base extensions. Suppose that the extensions of the classes `Library.Book` and `Project.Book` overlap. Since the integrity constraints of the class `Library.Book` are decomposable, they can be adopted to the base extensions 1 and 9. All decomposable constraints are combined conjunctively in the corresponding base extensions.
- When composing base extensions to global classes, we have to check whether the constraints are composable. An integrity constraint is called *composable* if it remains valid on any superset of the original extension. If a class is composed by the union of two base extensions which is not associated with any class constraints, then the corresponding object constraints of the base extensions can be disjunctively connected. However, base extensions usually contain class constraints such as uniqueness constraints. In this case, the problem is solved by introducing *discriminant*-attributes that are used to assign objects of the unified classes to the corresponding base extensions [6].

By integrating integrity constraints we can solve the global understandability problem. For solving the local understandability problem, the integration of integrity constraints must be done in a consistent way with respect to extensional assertions. However, since the extensional assertions are defined by the database integrator, the extensional assertions and the underlying local integrity constraints may be conflicting. Based on the idea that integrity constraints restrict the possible extensions of related classes, the derivation of extensional relationships can be partly supported as shown in [27] and Section 4.

5.3 Authorization Integration

A critical feature of database systems concerns the management of confidential data. Powerful mechanisms are needed for user management, authorization and authentication. In a federated environment, the global and local mechanisms have to work together to ensure global security. *Security* is achieved when the requirements *secrecy*, *integrity*, and *availability* are ensured.

Our focus lies on access rights. Access rights are statements about the rights of subjects (users, roles, programs, etc) to access objects (pieces of data) in certain ways (read, write, etc) including the right to grant them to other subjects [5]. In a federated system, local access rights have to be transformed, filtered and

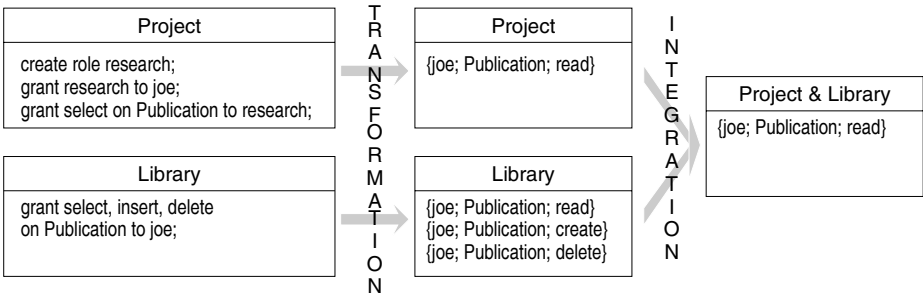


Fig. 6. Transformation and Integration of Access Rights

integrated. This includes the resolution of conflicts among overlapping access rights imported from different local systems.

During the design process of a federated system a global security model has to be chosen for expressing the integrated access rights. We express all access rights as triples consisting of the following concepts:

- *subjects*: only single users;
- *objects*: classes, objects, and attributes;
- *access types*: read, write, create, delete.

We have only positive access rights and we follow a closed world policy. This model can be extended with more complex concepts like user role models.

Fig. 6 illustrates a scenario with two local databases **Project** and **Library**. The local user named **joe** has read access to the class **Project.Publication** over the role **research**. In the other system, the local user **joe** may read and write the class **Library.Publication**.

At first, the access rights on the local schemata are transformed into access rights expressed in terms of our global model. Because of the simplicity of our model, we have to divide access rights based on complex concepts like for example groups of users or views which in turn increases the number of access rights significantly. In our example this affects the role **research** to which **joe** is assigned.

Afterwards, the exported local access rights are integrated into global access rights which correspond to the integrated schema. A semantic conflict may arise if a user has the right to access a certain data object in one local system but not the same or overlapping data object in another local system. Consider again Fig. 6. During schema integration the local classes **Project.Publication** and **Library.Publication** are integrated to the global class **Publication** (cf. Fig. 5). We integrate the users from the different local systems with a special user management tool that imports the local user profiles and ensures that all identical users are assigned with one global identity. In the case of our example we create the global subject **joe** which is joined with its two local counterparts. So, a conflict arises because **joe** has only read access to the corresponding local class **Project.Publication** and complete access rights to the other local class.

The designer of the federated system has the following three possibilities to resolve such a conflict in cooperation with the local administrators:

- The access is denied if this is required by the relevant administrator.
- The access is permitted if this is accepted by the relevant administrator.
- The right is locally granted. The conflict vanishes.

In our example we choose the first option. Thus, the user `joe` is only allowed to update `Library.Publication` at the local system.

Finally, we have to discuss how we use the integrated access rights within the scope of a federated authorization process. In the tool **SIGMA**_{Bench} we implement a variant of global authorization that checks all global queries concerning federated data against the integrated access rights. At the local systems, we establish a user with the login name `fdfs` to whom the responsible administrator grants full access to all data that can be exported. To obtain the necessary trust of the local systems for this approach, we additionally need powerful user authentication mechanisms at the global side [11].

6 Global Transactions

In a federated database environment, *global transactions* are able to transparently access and manipulate data located in different local database systems [4]. Depending on the underlying global schema, and particularly on the fixed extensional assertions, a global transaction is decomposed into a set of *global subtransactions* which operate on the local database systems. Extensional assertions influence the dependencies between the global subtransactions of the same global transaction [29]. Different *global commit rules* are needed to correctly terminate a global transaction. For instance, in some cases a commit of only one global subtransaction is sufficient to commit the global transaction, i.e., a global transaction can commit although some of the global subtransactions abort.

In the following, we consider the decomposition of a global transaction t_g and the corresponding global commit rule for this transaction. The global transaction t_g consists of the following insert on the global classes C_{g_1} and C_{g_2} (where the classes C_1 and C_3 are managed in one local database system while the classes C_2 and C_4 are managed in another local database system):

Insert into $C_{g_1} = C_1 \cup C_2$ with $C_1 \not\supset C_2$;
 Insert into $C_{g_2} = C_3 \cup C_4$ with $C_3 \supset C_4$;

The insertion into class C_{g_1} is successful if the corresponding object is inserted in exactly one of the local classes C_1 and C_2 . In contrast, an object is inserted into class C_{g_2} if the object appears at least in class C_3 . The global transaction is successful if either an object is inserted in class C_1 and class C_3 , in class C_2 and class C_3 , in the classes C_1 , C_3 , and C_4 , or in the classes C_2 , C_3 , and C_4 .

If only one of the operations of a transaction aborts then the whole transaction aborts. Thus, the global transaction is decomposed into global subtransactions such that transaction t_i performs the insertion into class C_3 , transaction t_j

into class C_1 , transaction t_k into class C_2 and transaction t_m into C_4 . The resulting dependencies are as follows (cf. termination dependencies in [23]):

- The abortion of transaction t_g requires the abortion of all subtransactions: $a_{t_g} \Rightarrow (a_{t_i} \wedge a_{t_j} \wedge a_{t_k} \wedge a_{t_m})$.
- The commit of transaction t_i is essential for transaction t_g : $(a_{t_i} \Rightarrow a_{t_g})$.
- If the transactions t_j and t_k abort, then the global transaction t_g has to abort, too: $(a_{t_j} \wedge a_{t_k}) \Rightarrow a_{t_g}$.
- Transaction t_j and t_k are in an exclusive dependency: $(c_{t_j} \Rightarrow a_{t_k} \wedge c_{t_k} \Rightarrow a_{t_j})$.
- The success of transaction t_m is optional.

The previous example shows that the assumption in the literature that at least one global subtransaction of the same global transaction is executed at the corresponding local systems, e.g., [4,10], cannot hold since local integrity constraints are not known and considered on the global level. The tool SPECTRAC enables us to specify arbitrary sets of transactions on the global level. These are connected over transaction dependencies like termination dependencies.

7 Conclusions

Building and maintaining federated database systems is a hard task which cannot completely be done in an automatic way. Our main goal is to support the schema designer of a federated database during the design process. We work on different areas of this complex problem and develop a bunch of solutions. The online documentation as well as a more detailed technical report of the **SIGMA**_{Bench} can be found under:

<http://wwwiti.cs.uni-magdeburg.de/~sigmafdb/>

From a scientific point of view, there is still a lot of open problems in database integration. One main problem is the extraction of semantic information from existing systems. Our approach has done first steps in analyzing file structures, database schemata, and existing explicit integrity constraints. However, schema re-engineering and extraction of implicit integrity constraints can never be completely automated and, thus, always require human interaction.

Besides the concrete federation aspects shortly described in this paper there is still a large number of other facets where methods and tool support are needed. Examples are integration of stored procedures, other databases models, general behavior integration, etc. Our aim is to build an open tool environment where developments in these areas can be ‘plugged in’ to enrich the design support offered by our tool.

References

1. A. Adelberg. NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In L. Haas, A. Tiwary, eds., *Proc. SIGMOD'98*, ACM SIGMOD Record, Vol. 25, pp. 283–294, ACM Press, 1998. 449
2. S. Balko, C. Türker. Integration of Aggregation Constraints. In S. Conrad, W. Has-selbring, G. Saake, eds., *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99*, pp. 5–24. infix-Verlag, 1999. 454, 455
3. C. Batini, M. Lenzerini, S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986. 446, 451
4. Y. Breitbart, H. Garcia-Molina, A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, 1992. 457, 458
5. S. Castano, M. G. Fugini, G. Martella, P. Samarati. *Database Security*. ACM Press, Addison-Wesley, 1995. 455
6. S. Conrad, I. Schmitt, C. Türker. Considering Integrity Constraints During Federated Database Design. In S. M. Embury, N. J. Fiddian, A. W. Gray, A. C. Jones, eds., *Advances in Databases, BNCOD 16*, LNCS 1405, pp. 119–133. Springer, 1998. 454, 455
7. U. Dayal, H. Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Transactions on Software Engineering*, 10(6):628–644, 1984. 454
8. Y. Dupont, S. Spaccapietra. Schema Integration Engineering in Cooperative Databases Systems. In K. Yetongnon, S. Hariri, eds., *Proc. 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems, PDCS'96*, pp. 759–765. International Society for Computers and Their Application, 1996. 454
9. B. Ganter, R. Wille. *Formal Concept Analysis*. Springer, 1998. 452, 453
10. D. Georgakopoulos, M. Rusinkiewicz, A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):166–180, 1994. 458
11. E. Hildebrandt, G. Saake. User Authentication in Multidatabase Systems. In R. Wagner, ed., *Proc. 9th DEXA Workshop*, pp. 281–286. IEEE CS Press, 1998. 457
12. R. Himmelröder, B. Ludäscher. Querying the Web with FLORID. In M. H. Scholl, H. Riedel, T. Grust, and D. Gluche, eds., *10. Workshop “Grundlagen von Datenbanken”*, Forschungsbericht No. 63, pp. 94–98, Universität Konstanz, Fachbereich Informatik, 1998. 450
13. M. Höding. An Approach to Integration of File Based Systems into Database Federations. In *Heterogeneous Information Management, Proc. 10th ERCIM Database Research Group Workshop*, pp. 61–71, ERCIM-96-W003, European Research Consortium for Informatics and Mathematics, 1996. 449
14. M. Höding, R. Hofestädt, G. Saake, U. Scholz. Schema Derivation for WWW Information Sources and their Integration with Databases in Bioinformatics. In [16], pp. 296–304. 446
15. J. A. Larson, S. B. Navathe, R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989. 452, 453
16. W. Litwin, T. Morzy, G. Vossen, eds., *Proc. ADBIS'98*, LNCS 1475, Springer, 1998. 459, 460, 460

17. A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13(7):785–798, 1987. 454
18. K.-U. Sattler, M. Höding. Adapter Generation for Extraction and Querying Data from Web Sources. In *Proc. 2nd ACM SIGMOD Workshop WebDB’99*, 1999. 450, 450
19. I. Schmitt, G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In B. Thalheim, ed., *Conceptual Modelling — ER’96*, LNCS 1157, pp. 195–210. Springer, 1996.
20. I. Schmitt, G. Saake. Merging Inheritance Hierarchies for Database Integration. In M. Halper, ed., *Proc. CoopIS’98*, pp. 322–331. IEEE CS Press, 1998. 446, 451, 453
21. I. Schmitt, G. Saake. Integrating Schemata using the GIM Method. Informatik Preprint, Universität Magdeburg, 1999. 451
22. I. Schmitt, C. Türker. Refining Extensional Relationships and Existence Requirements for Incremental Schema Integration. In G. Gardarin, J. French, N. Pissinou, K. Makki, L. Bougamin, eds., *Proc. CIKM’98*, pp. 322–330. ACM Press, 1998. 452
23. K. Schwarz, C. Türker, G. Saake. Extending Transaction Closures by N-ary Termination Dependencies. In [16], pp. 131–142. 458
24. A. P. Sheth, J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990. 445, 446, 451
25. S. Spaccapietra, C. Parent, Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, 1992. 451, 452
26. K. Schwarz, I. Schmitt, C. Türker, M. Höding, E. Hildebrandt, S. Balko, S. Conrad, G. Saake. Tool Support for the Design of Database Federations in SIGMA_{FDB}. Informatik Preprint 14, Universität Magdeburg, 1999. 445
27. C. Türker, G. Saake. Deriving Relationships between Integrity Constraints for Schema Comparison. In [16], pp. 188–199. 446, 450, 451, 455
28. C. Türker, G. Saake. Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration. In *Proc. ADBIS’99*, LNCS. Springer, 1999. 454
29. C. Türker, K. Schwarz, G. Saake. Commit Protocols for Global Transactions in Federated Database Systems. Informatik Preprint 2, Universität Magdeburg, 1999. 457

Operational Characterization of Genre in Literary and Real-Life Domains

Antonio L. Furtado and Angelo E. M. Ciarlini

Departamento de Informática
Pontifícia Universidade Católica do R.J.
22.453-900 Rio de Janeiro, Brasil
furtado@inf.puc-rio.br, angelo@inf.puc-rio.br

Abstract. Literary or real-life narratives are classifiable according to genres. In this paper, genre is used as a modelling concept, being characterized operationally, in terms of which facts and actions are allowed and the goals and interactions of the agents involved in a narrative pertaining to the genre. Intuitively, characterizing a genre means developing an executable specification capable of recognizing and/or generating plots of acceptable narratives. The specification is, in turn, extracted from an analysis of behavioural patterns. The machinery of multistage interactive plot generation is described and illustrated by way of running examples. A prototype, developed in Prolog extended with Constraint Programming, is being used to perform experiments both in literary and business contexts.

1. Introduction

Novels, sagas, fairytales and comedies are examples of *literary genres*. Swales [31, 26] demonstrates the existence, as well, of non-literary *real-life genres*, such as sets of instructions, letters of complaint, research papers, etc. Arguably, employee career records, student records, checking and saving account transactions, and so forth, are genres associated with widely-known database application domains. Cases in general (e.g. in jurisprudence and in management activities) constitute yet another genre of major relevance, having deserved attention both from a literary [16] and from an utilitarian perspective, especially with the present vogue of *case-based reasoning* [19]. In [14], we argued that temporal databases are, in general, repositories of narratives about the agents and objects involved. Such narratives may be classified into genres – the question here is how to perform the classification so that genre becomes a modelling concept of practical consequence. Our proposal is to favour an *operational approach*, along lines to be explained in the present paper.

When we examine a set of narratives of a certain literary genre, or even descriptions of real events in some practical application domain, we readily perceive that such occurrences reveal identifiable typical events and patterns. Furthermore, the events are not fortuitous. They are brought about by the *goals* of the *characters* (or *agents*) participating in the narratives.

On the basis of these remarks, and pursuing the multidisciplinary approach outlined in [14], we propose to study the automatic generation of *plots of narratives*,

aiming at: the operational characterization of both literary and real-life genres; supporting creation of new narratives of literary or practical interest; and supporting decision-making, particularly in business environments.

An advantage of operational over definitional or denotational characterizations [9, 13, 22] is that one gains, as a by-product of specifying a genre, a machinery for performing multiple kinds of experiments, including the two supporting tasks listed above. Our proposed method for plot generation – fully implemented in a prototype employing logic and constraint programming – consists of the combined use of two processes: the recognition of typical operational patterns and the simulation of the behaviour of the various agents. The generation is semi-automatic, enabling the user to interact continuously with the system. The user can interfere in three different ways: specifying and, later, modifying the context; supplying *observations*, i.e., conditions regarding the state of the agents and events that should be part of the plot; and choosing, among plot alternatives being developed, those which seem more promising.

The context for plot generation comprises a database, introducing the agents and their initial situation; a library of typical plans, establishing the admissible repertoire of events and plots, through the definition of operations and integrity constraints; and a set of logical rules, to infer goals to be pursued by each agent, as certain situations arise in the course of plots. The last two items characterize the genre, whereas the database provides the starting configuration, from which plots can be developed. The specification of the context is made in terms of a temporal logic compatible with the recognition and simulation processes.

In section 2, we indicate how concepts from Literary Theory are combined with techniques from Artificial Intelligence and Databases to provide the background for treating plots of narratives. Section 3 describes the prototype (*Interactive Plot Generator* - IPG) implemented for composing plots along a multistage interactive process. As evidence that the method and the prototype are versatile and have already reached a fair degree of expressive power, running examples are supplied in section 4, one in a business application domain involving the relationships of a company with its employees, and the other in a literary context. Section 5 contains concluding remarks and aspects needing further research.

2. Plots of Narratives

Fundamental to our research project was the seminal work of the Russian scholar Vladimir Propp [24], one of the forerunners of Structuralism, a school of literary theory whose influence extended to several other areas, such as Anthropology. Writing during the first half of the century, Propp objected to the traditional classification of genres according to *motifs*. An example of folktale motif is "love like salt" (n°923 of the Aarne-Thompson index [2]), where three sisters are called to demonstrate to their father which one loves him more, made popular through Shakespeare's adaptation in *King Lear*. Propp argued that, being decomposable, motifs are inadequate as basic units for classification purposes.

He proposed instead to extract *functions* from relatively short passages, associated with significant actions taking place in the narratives. In this guise, plots of narratives

might be described in terms of sequences of executions of these functions. Working over a large corpus of Russian fairytales [1], Propp identified 31 typical functions characteristic of this specific genre, such as *villainy*, *struggle of the hero against the villain*, *claims of a false hero*, etc. He admitted that not all functions needed to occur in every fairytale but, whenever present, their appearance invariably obeyed a single fixed order. However, a number of his own examples show that some freedom remains, which took us to view plots in general as *partially ordered sets* (posets) of function executions, rather than strict sequences. Continuing Propp's studies, Greimas [15] found it useful to reduce the number of functions for the fairytale genre, utilizing grouping criteria. Accordingly, for each genre or application domain envisaged, we decided to establish a *composition* hierarchy. Furthermore, already in [24], for every fairytale function several cases are enumerated. *Villainy*, for instance, can take the form of *kidnapping*, *mutilation*, *murder*, etc.; the *hero's reward* may be *wedding*, *accession to the throne*, etc. This, in turn, led us to also impose a *specialization* hierarchy.

Theories coming after Structuralism criticized what was considered to be an exclusively syntactic approach, failing to supply semantic grounds for, among other things, justifying the acceptable ways of chaining function executions. Even so, unbiased researchers recognize the value of the contributions of Structuralism. In particular, they lend themselves well to automatic processing; Propp's ideas, for instance, guided Rumelhart to introduce his "story grammar" formalism [27].

To add semantic contents to functions, we define them as *operations*, with *pre-conditions* and *post-conditions*, in the style of a STRIPS-like formalism [12]. *Villainy*, to give one example, has as pre-condition the fact that the *victim* is *vulnerable* and as post-condition the fact that the *victim* passes to a condition which causes *suffering*. The chaining of functions is naturally entailed by the interplay of pre-conditions and post-conditions. In order that the *victim* become *vulnerable* prior to the successful execution of *villainy*, some other event (i.e. execution of the corresponding operation) may be called for, such as *interdiction violated* or *death of parents*. Besides syntax and semantics, a *pragmatic* aspect must be modelled, which corresponds to the *goals* motivating the execution of an entire partially-ordered set of operations. Once their global primary effects are identified, these posets can be seen as *plans* of agents towards the achievement of their goals. A plot consists, then, of a set of executions (successfully completed or attempted) of plans. When we thus characterize plots of narratives, we gain the ability to handle existing plots and create new ones through *plan-recognition* and *plan-generation* techniques.

Whereas plan generation consists of creating one or more alternative plans able to achieve the goals specified, plan recognition tries to find out what plan an agent intends to execute by matching his observed actions against a library of typical plans. Each typical plan is a complex operation, recursively decomposable into simpler ones, so that the library is structured in the form of hierarchies of *composition* (part-of links) and *specialization* (is-a links). Fig. 1 shows a library of typical plans for a business environment, where a company deals with its employees and clients. Part-of links are denoted by single edges and is-a links by double edges. The specializations of the *end* node that do not have specializations are denominated *end-plans*. Once the user requests that a typical plan be detected wherein certain given events occur, the recognition algorithm starts looking for one or more end-plans containing all the indicated events.

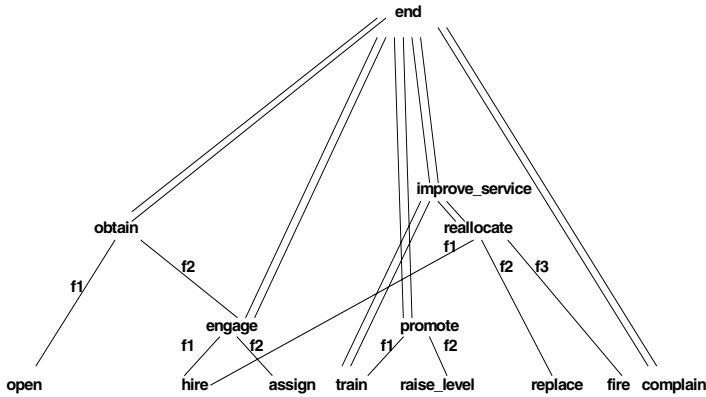


Fig. 1. Library of typical plans

It is worth expressing the pre-conditions and post-conditions of operations, as well as the initial state and the goals of the agents, in terms of *predicates*. To express, in a fairytale, that "the princess is suffering", we assert the *fact* (ground predicate instance) *victim_state(princess, suffering)*. Similarly, we may specify, in a business environment, that an employee *E* is at the service of a client *C* through the fact *serves(E,C)*. The initial situation of each agent is described by a database containing the respective predicate instances holding initially. As usual in the context of Deductive Databases, we adopt the *Closed World Assumption (CWA)* [25], according to which facts not contained in the database (nor deducible from other database facts) are considered false.

The execution of an operation changes the current database state, adding or removing facts. Thus, from a totally ordered sequence of events (executions of operations) results a temporal database [23]. Temporal databases enable the keeping of descriptions of all states reached, without making explicit, however, what actions caused the transitions. This sort of information becomes available if, in addition to the records of facts, a *log* registering the (time-stamped) executions of operations is maintained [11]. We will then say that, besides static descriptions, databases thus enhanced now contain *narratives*. If the order of events is only partial, one has a family of *possible* temporal databases. In order to reason about the validity of facts at given instants of time, one needs a *temporal logic*, able to handle both *valid facts* and *possibly valid facts*. In particular, this distinction is vital if we want to infer new goals of agents on the basis of still incomplete plots.

Having characterized plot generation as a database process, we are entitled to utilize concepts and techniques current in this area, especially those related to *temporal*, *deductive* and *active* databases.

To generate plots of narratives, one must preliminarily take into account the interactions among the motivating goals. Wilensky [33], dealing with plan recognition and generation in everyday life situations, classified the interactions among goals of one or more agents into *positive interactions*, where reaching a goal can help achieve other goals; and *negative interactions*, where reaching a goal can interfere, and even render impossible (totally or partially) the satisfaction of other goals. Negative

interactions comprise *conflicts* (referring to goals of the same agent) and *competitions* (related to goals of diverse agents).

Studying the mental act of understanding stories, Wilensky [32], argued that, to raise the reader's interest, stories must have *points*. Points are mostly negative goal interactions which precipitate a series of events in a not too obvious fashion. Thus, plot generation must include mechanisms supporting not only positive but, more importantly, also negative interactions. This creates the need to handle cases of *abandonment of goals* (often necessary to solve conflicts) and of *competitive plan execution*, wherein goals of one agent may be frustrated in the benefit of those of another (to solve competitions). It is not only in literary composition that negative interactions play a fundamental role. Simulations for decision support, for instance, tend to prove more useful in situations involving critical conflicts among agents.

Our model uses plan-recognition and plan-generation as complementary processes in the generation and understanding of interactions of database agents. For a complete model, we basically need four kinds of transformations:

1. Plan recognition: observations $\rightarrow$ possible plans. Observations about database states and operations thus far attempted or actually executed are matched against a library of typical plans, to predict which known typical plans might be intended by the agents.
2. Goal recognition: plan $\rightarrow$ goal. Once a plan is recognized, through the matching process above, its corresponding goal is determined as a by-product of the process, since each typical plan kept in the library is a complex operation the main effect of which is registered as part of its definition.
3. Plan generation: goal $\rightarrow$ plan. Given a goal, a planner is applied for the generation of partially-ordered sets of operations able to achieve the goal.
4. Goal generation: database state $\rightarrow$ goal. The execution of a plan changes the current database state, and, as a result, further goals can be inferred by activating appropriate rules relating the new situation with the needs and motivations of the various agents.

Transformations 1 and 2 are directly related to the recognition of patterns, whilst transformations 3 and 4 have to do with the simulation process. Our treatment of simulation, adopts the viewpoint of an *author*, who keeps an attitude of neutrality towards the characters. Their behaviour is modelled through goal-inference rules characteristic of the genre. Once goals are inferred taking into account the problems and opportunities emerging at the current state, possible events that these goals may bring about are considered, together with the possible resulting interactions among the agents. When an agent executes operations to achieve one of his goals, he changes the current database state. New goals, both of the original agent and of others, may arise as a result of the modification. Further executions of operations, corresponding to new plans, are necessary to achieve the newly identified goals. Thus, we have a progressive process, in which a plot (denoting, as explained, a database narrative) is generated on the fly by the alternation between generating goals and planning additional executions of operations. It is important to notice that new goals are motivated not only by past events but also by operations already scheduled for future execution and by the beliefs of each agent about the future behaviour of the others.

Another crucial feature of our approach is that our plots are not restricted to only incorporate successful plans. As we endeavoured to provide adequate means to handle negative interactions along plots, we realized that the solution of conflicts and competitions sometimes requires the inclusion of totally or partially *failed* plans, which conventional plan generators automatically reject. When a goal is abandoned, events occurring before the time of abandonment must be kept as part of the narrative, and therefore must cause their effects in how the narrative is developed.

hints about the plot to be generated (some of the operations to be executed, a few assertions about the situation of agents at intermediate states and goals to be finally reached, etc.), and the system looks for a typical plan pattern (or set of patterns) that may fit the observations supplied. The system fills in what is missing in the characterization of agents, taking into consideration the observations and the pertinent facts found in the initial state of the database. The second form, in contrast, is based exclusively on the Simulator. The user enters observations about the situation he wishes to simulate; then, the rules specifying the behaviour of agents are evaluated over these observations and the initial state of the database, in an attempt to infer new goals. From this point on, alternate phases occur of plan generation (to achieve such goals) and of inference of goals (at each state reachable by the plans), until no further goals result. The third form involves an integrated use of the Simulator and the Recognizer. Upon recognition of a plot P, some of the pre-conditions of its constituent operations may not hold in the database. One option in obtaining a valid plot is to adapt P, with the help of the Simulator. An alternative is to simulate an entirely new plot, whereby the primary effects of P are achieved. Even if a recognized plot is totally valid (all its pre-conditions hold), there is still the possibility of extending it with plans induced by the goals inferred at states determined by their effects. Such extensions can either be proper continuations, added strictly after the completion of the plot, or plot enrichments, with the additional sequences of operations interpolated or ranged in parallel.

The Recognizer uses a plan-recognition algorithm adapted from the algorithm defined by Kautz [17]. The adaptations extend it to handle pre- and post-conditions of operations and establish a connection with the Simulator. The Recognizer identifies patterns by matching the observations supplied against the library of typical plans (defined as described in section 2). Observations concerning states reachable during the execution of plots are verified by consulting the Temporal Logic Processor. The plots recognized are communicated to the user, together with the indication that its pre-conditions can (or cannot) be satisfied. According to the user's choice, the Simulator can be activated to adapt or extend plots, or else to generate other plots with the same primary effects identified. The Recognizer explores all alternatives recognizable and displays them to the user, as requested.

The Simulator executes planning tasks, supported when necessary by the Temporal Logic Processor. To obtain goals, the Simulator interacts with the Goal Evaluator in a multi-stage process. During simulation, the selection, from the set of partially generated plots, of those still deserving examination is made by interacting with the user. When a partial plot is exhibited, the system signals to the user that the Query Interface is available. The Query Interface enables the user to formulate queries about the situation of agents at diverse states during the plot. For this purpose, the Query Interface can also access the Temporal Logic Processor. After consulting the Query Interface, the user can return normally to the Simulator and decide whether or not to proceed with the current plot.

In our examples, the construction of the library of typical plans was done manually. The same happened with behaviour rules, in the formulation of which we were guided by previous work on goal relationships and interconnections of events. We are now investigating methods for supporting these processes, to become part, respectively, of the future *Library Construction* and *Rule Formulation* modules.

The IPG prototype was implemented in SICSTUS Prolog [6], and extensively employs its constraint programming features to express arithmetic relationships and non-codesignations among variables, and also to evaluate temporal logic expressions.

To support the processes of pattern recognition, planning and goal inferencing, one needs to define a logic enabling the expression of and reasoning about the actions that modify the database and about the goals that motivate the agents to perform these actions. To characterize modification of states by means of operations, a Temporal Logic notation, e.g. Event Calculus [18], would seem to be an obvious choice. However, as we are working with database events, we decided to also borrow from Multi-Sorted Database Logic [3]. Accordingly, our logic is a Multi-Sorted Database Logic in which a special status is conferred on the sort *time-stamp*. Our truth model is based on the *Modal Truth Criterion (MTC)* [5], which is fully compatible with Event Calculus with Preconditions [4].

According to MTC, a condition *C* is *necessarily true* at the execution time *t*₁ of an operation (denominated *user*) if *C* was necessarily established by another operation (*establisher*) at time *t*₂, before *t*₁, and there is no operation (*clobberer*) that can establish the negation of *C* at a time *t*₃ between *t*₂ and *t*₁. A condition is *possibly true*, at a given time *t*₁, if it is either necessarily true or can be made true by the addition of constraints on time and on the values of variables.

Our temporal logic is used in the characterization of the genre, in particular for specifying integrity constraints, pre-conditions and post-conditions of operations, and rules for the inference of goals. It is also used to enable the user to formulate queries about the plots being generated, as well as to inform the system about the observations to be taken into account, either for pattern-recognition or in simulation runs. We do not formalize our logic here, due to space considerations. Instead, we describe informally the meta-predicates used for speaking about database facts and operations associated with time. We then demonstrate the use of our notation indicating how to write rules, using these meta-predicates, aiming at the specification of the behaviour of agents.

The order of execution of operations being merely partial, two different criteria for truth evaluation are needed. We should be able to verify whether a fact associated with a certain time-stamp is either necessarily or possibly true. In addition, it must be possible to speak about the time when a fact was established by an operation, because a condition is valid only after (but not at) its establishment. We should also be able to mention the time when a specific operation happened (i.e. was executed). Accordingly, we have four temporal meta-predicates, noting that *LITERAL* stands for a positive or negative fact: *h(T,LITERAL)*: *LITERAL* is necessarily true at *T*; *p(T,LITERAL)*: *LITERAL* is possibly true at *T*; *e(T,LITERAL)*: *LITERAL* is established at *T*; and *o(T,OPERATION)*: *OPERATION* happened at *T*. In order to express constraints relating variables of specific sorts, we have two additional meta-predicates: *h(CONSTRAINT)*: *CONSTRAINT* is necessarily true; and *p(CONSTRAINT)*: *CONSTRAINT* is possibly true.

The atomic formulae are meta-predicates and negations thereof. The rules are formulated as implications, in which the antecedent is a conjunction of atomic formulae and the consequent is a conjunction of meta-predicates of types 1 and 5 only. Furthermore, whenever conditional or limited goals are involved, their associated conditions or limits, respectively, must be included in the rule definition. If the antecedent holds, then the consequent will correspond to a goal to be pursued (unless it already holds). By assumption, all variables in the antecedent are

universally quantified, whereas those appearing only in the consequent are existentially quantified. For consistency with the Closed World Assumption, variables that appear only in negated meta-predicates or facts are regarded as existentially quantified within the scope of the negation. Variables are represented by capital letters. “_” stands, as in Prolog, for anonymous variables. In our database example, to express the rule “If *C* became a client at time *T1* and at this time no employee is assigned to *C* and there is a person *E* who is not serving any client, then the company will try to arrange that, at a time *T2*, after *T1*, *E* will be serving *C*” we write:

$$e(T1, is\_client(C)) \wedge \neg e(T1, serves(\_ , C)) \wedge h(T1, person(E)) \wedge \\ \neg h(T1, serves(E, \_)) \rightarrow h(T2 > T1) \wedge h(T2, serves(E, C))$$

The pattern-recognition process executed by the Recognizer receives input *observations*, supplied by the user, including events that must occur as part of the plots, goals to be achieved at the end of the plots, mandatory intermediate states, and restrictions over data or time stamps (used to reference the instants at which events occur or intermediate state facts must hold). The expected output is a set of typical plans/plots compatible with the observations and the initial database state.

An example, in the context of a corporate database, is the detection of the typical plot *employee's promotion*, being given, as observation, the fact that *a certain employee took a certain training course*. Thus, the typical plot recognized would include, besides the event *take course*, other components of the typical plot, such as *apply for salary raise* and *receive salary raise*. If the course to be taken by the employee is not given, the Recognizer should be able to select one in conformity with the typical *employee's promotion* plot and the facts holding at the current database.

As the Recognizer processes a given observation, it identifies the events mentioned in it, and tries to build explanation graphs showing how all these events can be part of the same end-plan. Then, for each end-plan recognized, it checks if its pre-conditions are satisfied at the initial state of the database. It also checks that it does not violate any integrity constraint and whether the remaining conditions specified in the observations are satisfied. If any of these requirements fails, the end-plan is discarded. The end-plans successfully recognized are reported to the user, who decides whether the simulation process should proceed from the patterns proposed by the system.

Aiming at the simulation of database agent interactions, we use planning methods to discover possible operations that the agents can execute to achieve their goals. Planners implementing a simple backward-chaining strategy are not enough to solve our problem. They do not cover the case in which many goals of many different competing or collaborating agents are to be achieved and they introduce constraints that are not imposed by the satisfaction of goals but rather by the algorithm itself. Plans are treated as totally ordered sequences, so that, for all pairs of operations, one must precede the other for execution, even when there is no causal connection between them. There is no priority for the resolution of goals, and therefore it is not permitted to establish, for instance, that corporate goals have higher priority than any other goals. Moreover, conventional planners neither support abandonment of goals nor competitive plan execution, which usually entails the need to force one agent to give up one of his goals. Because of these reasons and also in the benefit of efficiency, we looked at *non-linear planners* in the Tweak [5] category.

A non-linear plan consists of a set of partially ordered operations. In order to check whether a pre-condition of an operation is satisfied, non-linear planners apply the MTC, explained before. They use a *least commitment* strategy, according to which a

constraint (on the order of execution of operations or prescribing codesignations and non-codesignations of variables) is activated during the generation of a plan only when necessary. This then happens when the constraint must be applied to consider the insertion of an operation for establishing a pre-condition of another operation, or to resolve conflicts between operations.

In order to be able to assign priorities for the resolution of certain goals, and also to enhance the performance of our simulation environment, we adapted a non-linear planner (AbTweak [34]) that is also a hierarchical planner [35]. It tries to resolve all pre-conditions of all operations at a certain level before taking into account lower level pre-conditions, which is convenient particularly when we envisage forcing an agent to give up one of his goals. Being inserted first, operations for solving a high priority goal may prevent the insertion of operations aiming at conflicting lower priority goals. That will be the case if we establish that “corporate goals are more important than goals of employees”.

The planner performs a heuristic search for good plans. It works on many plans in parallel and, at each time, selects the candidate with minimal estimated cost for achieving a complete solution, i.e. a configuration in which all goals and all preconditions of all operations are satisfied. It then selects a pre-condition of an operation that is not necessarily true and tries to make it true. While doing that, the planner generates all possible successors of the current plan. It considers the possibility of using establishers already in the plan as well as the insertion of new operations. All possibilities for solving conflicts caused by clobberers are considered.

We use two main mechanisms to simulate goal abandonment and competitive plan execution: *conditional goals* and *limited goals*. Conditional goals have a *survival condition* attached to them, which the planner must check to determine whether the goal should be pursued or not. Operations added for the fulfilment of conditional goals are also conditional. They are kept as part of the plan only if their survival conditions are valid at the appointed execution time. However, even in case of failure, the planner keeps the operations preceding the instant when the failing survival conditions ceased to hold. Survival conditions and pre-conditions must be handled in a consistent way: the planner begins by trying to solve all pre-conditions of all operations at least once; if a survival condition of an operation fails, the planner determines that its pre-conditions will not be considered for a second time. Limited goals are those that are tried once only, and have an associated *limit* (expressed as a natural number). The limit restricts the number of new operations that can be inserted to achieve the goal. An operation inserted for the establishment of limited goals is termed a limited operation. A limited operation tentatively inserted in a plan can be kept only if all its pre-conditions are true. In addition, all its pre-conditions are in turn limited subgoals, the limit of which is that of the original goal minus one. A goal with limit 0 can only be satisfied by an operation already present in the plan. We use limited goals to model situations in which agents are willing to invest no more than a certain measure of effort, not necessarily the same for each agent, towards the fulfilment of their goals.

In order to adapt planning methods to the database context, we implemented two additional features. We modified MTC to cope with the *Closed World Assumption*. Therefore, we had to provide a special treatment for the evaluation of negative conditions, because they can be established simply by observing the absence of the corresponding positive facts in the database. We also used *Constraint Logic Programming* [21] to deal with pre-conditions that include arithmetic constraints,

frequently involved in database updates. The added features enable the handling of such pre-conditions even in the presence of free variables, since Constraint Logic Programming leaves a goal frozen until the moment it can be evaluated. In order to process numbers, it provides a solver of systems of equations and inequations able to run in parallel with the planning procedure. The solver tries to find out as early as possible whether a frozen goal can no longer be fulfilled, in which case it immediately cuts the current branch of the search tree, thus causing an improvement in the performance of the overall system.

When there is a set of goals to be achieved, by possibly different agents at possibly different times, the simulation performs the planning task. When all these currently active goals have been either fulfilled or discarded, the planning process completes one stage. The set of existing rules is then exhaustively evaluated, over the initial database state and the alternative future states resulting from the set of partially ordered operations generated thus far. If the consequent of a rule does not already hold and the corresponding goals have not previously been tried, these new goals will be pursued in the next stage of the planning process. The cycle ends when no new goal can be generated. This progressive generation of goals can be compared to the triggering process in *Active Databases* [7]. A trigger is usually a rule defined by a triple $\langle \text{Event}, \text{Condition}, \text{Action} \rangle$. When an event happens, a condition is evaluated and, if the condition is true, the corresponding action is performed automatically. Trigger events and conditions are semantically equivalent to the antecedents of our rules. The consequent of a trigger, however, is an action, whereas the consequent of our rules simply corresponds to goals.

4. Running Examples

In order to illustrate the use of our approach in practical situations, we created an example corresponding to the context of a company named Alpha. The library of typical plans is that shown in Fig. 1 with some new operations. In this example, there are three different classes of agents: employees, clients and company Alpha itself. In order to model their behaviour, we defined in our logic notation the following rules, here transcribed informally for easier reading:

1. Whenever a client C has no employee to serve it, and there are people still unassigned to any client, such people will compete for the position until one of them achieves the goal.
2. Whenever an employee is not content with his salary, he will try, in the measure of his persistence, to increase his status by one.
3. Whenever an employee has a status higher than his salary indicates, his salary level will be raised by one.
4. Whenever the salary of an employee E goes beyond the budget of client C, whom E is serving, E will not be permitted to continue serving C.
5. Whenever an employee E is not associated with any client and there are neither clients without employee nor potential clients to conquer, E will cease to be Alpha's employee.
6. Whenever a client C is dissatisfied with an employee E at its service, some action will be undertaken to remedy the situation.

Rules 1 and 2 give origin, respectively, to conditional and limited goals.

An initial database is given with 3 employees (David, Leonard and John), one unemployed person (Mary) and three clients (Beta, Epsilon and Lambda). David is serving Epsilon and Leonard is serving Lambda. All employees have salary level 1,

- *Kidnapping*: “A tsar has three daughters. The daughters go walking, and overstay in the garden. A dragon kidnaps them. The tsar calls for help. Three heroes fight the dragon and rescue the maidens. The heroes return and are rewarded”.
- *Expulsion by Water*: “Parents, afraid of a prophecy according to which their son would humiliate them, put the sleeping boy in a boat and set it adrift. Boatmen save the boy and take him to another land. Arriving there, he solves a task proposed by the tsar. He marries the tsar’s daughter; he travels home; en route he recognizes his parents at an overnight lodging place”.
- *Simple Murder*: “A sister, jealous of her brother, deceives and kills him. A thin reed grows upon the grave. A shepherd plucks it and makes a pipe out of it. The pipe plays and reveals the murderess. The parents expel the daughter”.

We created a database to represent the initial situation of each tale. We also imposed general rules about the motivation of fairy tales’ characters in specific situations. Once we set the database of each tale, IPG was able to generate, after some iterations, exactly the corresponding plot. Moreover, when we modified the initial database, IPG delivered new – and entirely coherent – plots. For instance, if we modify the initial database of *Kidnapping*, establishing a *strength_level* for the heroes lower than the *strength_level* of the dragon, *Kidnapping* is recognized but is not applicable. Then, if we ask IPG to adapt the plot, it inserts the new operation *out_of_the_earth_receipt*, which increases the heroes’ strength before victory. Here there is a parallelism: the new operation can take place at any time before the victory.

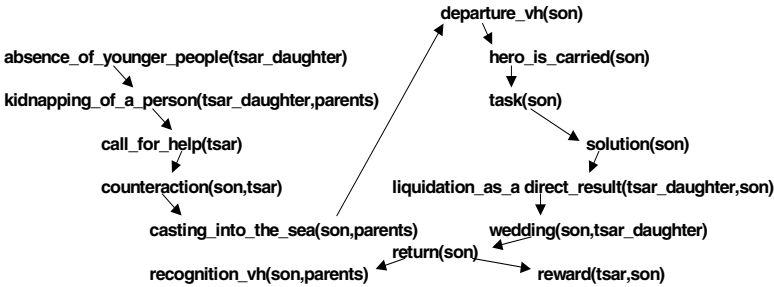


Fig. 4. Mixing Plots

Fig. 4 shows a plot obtained when we asked IPG to generate a new plot with the same primary effects of plot *Kidnapping*. The arrows indicate the order of the operations. The initial database was the database of plot *Expulsion by Water* expanded with some facts from the database of plot *Kidnapping*. The generated plot is a mix of the two plots: “The parents kidnap the tsar’s daughter; the tsar calls for help; the villains’ son agrees to help; the parents cast the boy into the sea; the boy is saved, fulfils a task, rescues the tsar’s daughter, marries her, returns, humiliates his parents and is rewarded by the tsar”.

5. Concluding Remarks

The present research confirms that a genre, such as the folktale genre, can be precisely characterized operationally, building on the contribution of Propp and other scholars and on Cognitive Science research. Characterizing a different genre – be it literary or real-life – involves passing to a different repertoire of predicates,

operations, and rules. Literary applications are useful to determine the possibilities and limitations of the approach. Their study may help in separate projects, involving retrieval over textual databases and comparative literary structural analysis.

In the realm of management information systems, IPG offers, through simulation, answers to "what if" questions, so crucially relevant to decision-making. A user conducting simulation experiments thus has an opportunity to anticipate and compare the favourable and unfavourable results of different lines of action, some of which may conform to the traditional attitudes and policies adopted in the past, while others would exploit valid but still untried alternatives. Our environment leaves room to the user's interference, so that his personal knowledge can guide the choice among options. Besides helping to decide with respect to specific situations, its continued use may disclose shortcomings in how agents and operations are currently understood, thus leading to the revision of the inadequate models.

The planning problem is known to be semi-decidable in general, and planning algorithms are, in the unrestricted cases, exponential. Therefore, good heuristics are indispensable to the speeding-up of convergence in the hierarchical planning process. We have been looking at this topic in the context of our prototype system, but further work is still needed. Even with the present version, however, promising results have already been obtained in very diverse domains, enabling the treatment of considerably more complex plots of narratives than commonplace database applications. As indicated, folktales are providing a nontrivial benchmark, and the schema of the narrative related to company Alpha suggests that phenomena involving "circular causality", with positive and negative feedback effects, such as the destabilizing effect of government efforts to control economic crises [20], can also be handled. Further research will involve gaining experience with our interactive multistage planning paradigm, as well as investigating methods to be applied in the enhancement of the system, with particular emphasis on the automatic construction of libraries of typical plans and formulation of behaviour rules.

References

1. A. Afanas'ev. *Russian Fairy Tales*. N. Guterman (trans.). Pantheon Books, 1945.
2. A. Aarne. *The Types of the Folktale: A Classification and Bibliography*. Translated and augmented by Stith Thompson, FF Communications, 184. Suomalainen Tiedekatemia, 1964.
3. M.A. Casanova and P. Bernstein. "A Formal System for Reasoning about Programs Accessing a Relational Database". *ACM Transactions on Programming Languages and Systems*. 2(3): 386-414, July 1980.
4. I. Cervasato, M. Franceschet and A. Montanari. "Modal Event Calculi with Preconditions". In *Proc. of the 4th. International Workshop on Temporal Representation and Reasoning*, 1997.
5. D. Chapman. "Planning for conjunctive goals". *Artificial Intelligence*, 32:333-377, 1987.
6. M. Carlsson and J. Widen. *Sicstus Prolog Users Manual, Release 3.0*. Swedish Institute of Computer Science, 1995.
7. S.Ceri and J. Widom. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.
8. Y. Demazeau. "From Interactions to Collective Behaviour in Agent Based Systems". In *Proceedings of the First European Conference on Cognitive Science*, 1995.

9. J. E. Donahue. *Complementary Definitions of Programming Language Semantics*. Springer-Verlag, 1976.
10. M.G. Dyer. In *Depth Understanding*. The MIT Press, 1983.
11. A. L. Furtado and M. A. Casanova. "Plan and schedule generation over temporal databases". In *Proc. of the 9th International Conference on the Entity-Relationship Approach*, 1990.
12. R. E. Fikes and N. J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence*, 2(3-4), 1971.
13. A. L. Furtado and E. J. Neuhold. *Formal Techniques for Data Base Design*. Springer Verlag, 1986.
14. A. L. Furtado. "Narratives and Temporal Databases: An Interdisciplinary Perspective". In *Conceptual Modeling: Current Issues and Future Directions*. P.P.Chen, J.Akoka, H.Kangassalo, B.Thalheim (eds.) Springer-Verlag, 1999.
15. A.J. Greimas. *Sémantique Structurale*. Librairie Larousse, 1966.
16. A. Jolles. *Formes Simples*. Éditions du Seuil, 1972.
17. H. A. Kautz. "A formal theory of plan recognition and its implementation". In *Reasoning about Plans*. J. F. Allen et al (eds.). Morgan Kaufmann, 1991.
18. R. Kowalski, M. Sergot. "A Logic-based Calculus of Events". *New Generation Computing*, 4: 67-95, Ohmsha Ltd and Springer-Verlag, 1986.
19. D. B. Leake (Ed.). *Case-based Reasoning: Experiences, Lessons, & Future Directions*. AAAI Press, 1996.
20. G. Morgan. *Images of Organization: the Executive Edition*. Berrett-Koehler Publishers, 1998.
21. K. Marriot and P.J. Stuckey. *Programming with Constraints*. MIT Press, 1998.
22. C-H. L. Ong. "Correspondence between Operational and Denotational Semantics". In *Handbook of Logic in Computer Science: Semantic Modelling*, Vol. 4. S. Abramsky, D. Gabbay and T. S. E. Maibaum (eds.). Clarendon Press, 1995.
23. G. Ozsoyoglu. and R.T. Snodgrass. "Temporal and real-time databases: a survey". *IEEE Transaction on Knowledge and Data Engineering*, 7, 4, 1995.
24. V. Propp. *Morphology of the Folktale*. Laurence Scott (trans.). Univ. of Texas Press, 1968.
25. R. Reiter. "On Closed World Databases". In *Logic and Databases*. H. Gallaire and J. Minker (eds.), Plenum Press, 1978, pp. 55-76.
26. S. A. Robertson. "The contribution of genre to computer supported cooperative work". In *Linguistic Concepts and Methods in CSCW*. J. H. Connolly and L. Pemberton (eds.). Springer-Verlag, 1996.
27. D.E. Rumelhart. "Notes on a schema for stories". In *Representation and Understanding - Studies in Cognitive Science*. D. G. Bobrow and A. Collins (eds.). Academic Press, 1975.
28. R.C. Schank and R.P. Abelson. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, 1977.
29. R.C. Schank. "Language and Memory". *Cognitive Science*, 4, 3, 1980.
30. R.C. Schank. "Reminding and Memory Organization: An Introduction to MOPs". In *Strategies for Natural Language Processing*. Lawrence Erlbaum Associates, 1982.
31. J. Swales. "A genre-based approach to language across the curriculum". In *Proc. of the RELC Conference*, 1986.
32. R. Wilensky. *Planning and Understanding*. Addison-Wesley Publishing Company, 1983.
33. R. Wilensky. "Points: a theory of the structure of stories in memory". In *Readings in Natural Language Processing*. B. J. Grosz, K. S. Jones and B. L. Webber (eds.), Morgan Kaufmann, 1986.
34. S.G. Woods. *An Implementation and Evaluation of a Hierarchical Non-linear Planner*. Master's thesis. Computer Science Department of the University of Waterloo, 1991.
35. Q. Yang; J. Tenenber; S. Woods. "Abstraction in nonlinear planning". Technical Report # CS-92-32, University of Waterloo, 1992.

Contextualization as an Abstraction Mechanism for Conceptual Modeling

Manos Theodorakis^{1,2}, Anastasia Analyti¹, Panos Constantopoulos^{1,2}, and
Nicolas Spyratos³

¹ Institute of Computer Science, FORTH, P.O.Box 1385, GR 711 10 Heraklion, Crete, Greece
{etheodor, analyti, panos}@ics.forth.gr

² Department of Computer Science, University of Crete, Heraklion, Greece
{etheodor, panos}@csd.uoi.gr

³ Universite de Paris-Sud, LRI-Bat 490, 91405 Orsay Cedex, France
spyratos@lri.fr

Abstract. The notion of context appears in several disciplines, including computer science, under various forms. In this paper, we are concerned with a notion of context in the area of conceptual modeling. First, we present a simple definition whereby a context is seen as a set of objects, within which each object has a set of names and possibly a reference: the reference of the object is another context which "hides" detailed information about the object. Then, we enhance our simple notion of context by structuring its contents through the traditional abstraction mechanisms, i.e. classification, generalization, and attribution. We show that, depending on the application, our notion of context can be used either as an alternative way of modeling or as a complement of the traditional abstraction mechanisms. Finally, we study the interactions between contextualization and the traditional abstraction mechanisms as well as the constraints that govern such interactions.

1 Introduction

The notion of context is of fundamental importance in cognitive psychology, linguistics, and computer science. In computer science, a number of formal or informal definitions of some notion of context have appeared in several areas, such as artificial intelligence [11,17], software development [24,10,26], [27,15], databases [3,9], [1,6,14,23], machine learning [18,16], and knowledge representation [21,33], [31,36,7,29], [5,4]. See also [19] for a general survey on the subject.

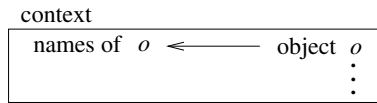
However, all these notions of context are very diverse and serve different purposes. In software development the notion of context appears in the form of views [3,9], [25,1], aspects [24], and roles [10,26], for dealing with data from different perspectives, or even in the form of workspaces which are used to support cooperative work [15]. In machine learning, context is treated as environmental information for concept classification [18,16]. In so called "multi bases", context appears as a collection of meta-attributes for capturing class semantics [14]. In artificial intelligence the notion of context appears as a means of partitioning knowledge into manageable sets [12], or as a logical construct that facilitates reasoning activities [17]. In particular, in the area of

knowledge representation, the notion of context appears as an abstraction mechanism for partitioning an information base into possibly overlapping parts [21,32,33], or for dividing the global schema of a database into clusters in order to deal with schema complexity [36,7,29,5,4].

Our objective in this paper is to establish a formal notion of context to support the development and effective use of large information bases in various application areas.

A *context* in an information base can be seen as a higher-order conceptual entity that groups together other conceptual entities from a particular standpoint. Contexts allow one to focus on the objects of interest, as well as to name each of these objects using one or more convenient names [22,32].

Roughly speaking, each object of a context is associated with a set of names as in the following diagram:



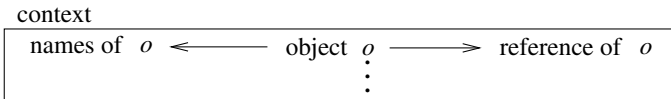
For example, in the context of a research group, the object o can be a researcher, with his social name (e.g., "John") and his nickname within the group (e.g., "The_Hacker") as two of its names.

It is important to note that the same object can belong to different contexts with different names in each of them. Thus, to the extent that names convey meaning, this is one way of providing context-dependent interpretation of objects. As we shall see, there are more ways in which the notion of context supports relative interpretation.

A context can be created on the basis of one or more criteria, such as: (i) *temporal*, e.g., the map of Greece through the centuries, where each century constitutes a different context; (ii) *spatial*, e.g., the economy of different regions of Greece, where each region constitutes a different context; (iii) *functional*, e.g., the usage of a knife in different human activities, where each activity constitutes a different context; (iv) *structural*, e.g., the organization of a computer from a hardware or from a software point of view, where each point of view constitutes a different context; and so on.

In this paper, we enhance the notion of context in two ways:

1. We allow each object of a context to be associated with another context that we call its *reference*. Thus, each object of a context is now associated with a set of names on one hand and (possibly) with a reference on the other, as in the following diagram:



Roughly speaking, the reference of the object points to information available about the object.

2. We allow the objects of a context to be structured through the traditional abstraction mechanisms of classification, generalization and attribution¹. We study how

¹ By "attribution" we mean the assignment of an intrinsic attribute to an object as well as the declaration of its (binary) relationships to other objects. The abstraction mechanism of *aggregation* is a limited form of attribution [13].

these three abstraction mechanisms interact with contextualization, in particular how instance-of, ISA, and attribute links between objects affect the definition of their references.

So, a context is a structured set of objects, in which each object is associated with a set of names and (possibly) a reference.

It is important to note that this notion of context allows to group together such things as class instances, classes, metaclasses, subclasses, superclasses, attributes, ISA links, and instance-of links.

In this paper, we only present the mechanism of context and the ways in which contexts interact with each other and with the traditional abstraction mechanisms of conceptual modeling. We do not consider methodological issues such as criteria for context formation.

The remainder of the paper is organized as follows: In Section 2, we define the notion of context *without* structuring of its objects, and we discuss some of its modeling capabilities. In Section 3, we enhance the notion of context by *adding* structure among the objects through the traditional abstraction mechanisms. In Section 4, we study the ways in which contexts interact with each other in the presence of the traditional abstraction mechanisms. In Section 5, we compare our framework with that of related works. Finally, in Section 6, we make some concluding remarks and suggestions for further research.

2 The Notion of Context

Suppose we want to talk about Greek islands by simply using their names without further description. Let us consider the island of Crete. We can represent this island by an *object identifier*, say o_1 , and by associating this identifier with the name Crete. We write $names(o_1) = \{\text{Crete}\}$ and we denote this as follows²:

Crete : o_1

Next, let us consider the island of Santorini. Following a similar approach, we represent this island by an object identifier o_2 and by associating it with the name Santorini. However, the island of Santorini is also known under the name Thera. So this time, we associate o_2 with the set of names $\{\text{Santorini}, \text{Thera}\}$, i.e., this time we write $names(o_2) = \{\text{Santorini}, \text{Thera}\}$ and we denote this as follows:

Santorini, Thera : o_2

Finally, let us consider one of those tiny, uninhabited islands of Greece that happen to be nameless. We represent such an island by an object identifier o_3 and by associating it with no name, i.e., we write $names(o_3) = \{\}$ and we denote this as follows:

: o_3

Continuing in the same way, we can represent every Greek island in a similar manner. The set of all such representations is what we call a *context* and we represent it by a *context identifier*, say c_1 , as shown in Figure 1.

Suppose next we want to talk about the Greek mainland by simply using the names of each region of Greece without further description. Proceeding in a similar way as in the case of Greek islands, we can create a second context, say c_2 , as shown in Figure 1.

² In this paper, the terms *object* and *object identifier* will be used interchangeably.

Suppose now that we want to talk about Greek geography seen as a division of Greece into islands and mainland. First, let us consider the islands. We can represent the islands by an object identifier, say o , and by associating it with the name `Islands`, i.e., $names(o) = \{\text{Islands}\}$. However, the object o is a higher level object that collectively represents all Greek islands, i.e., the object o collectively represents the contents of context c_1 . In other words, if we want to see what o means at a finer level of detail, then we have to “look into” the contents of c_1 . Thus we call context c_1 the *reference* of object o , and we write $ref(o) = c_1$. Summarizing our discussion on islands, we write $names(o) = \{\text{Islands}\}$ and $ref(o) = c_1$, and we denote this as follows:

$$\text{Islands} : o \cdots \triangleright c_1$$

Following a similar reasoning, we can represent the mainland by an object identifier, say o' , and by associating it with the name `Mainland` and the reference c_2 . We can now group together the islands and the mainland to form a context c , as shown in Figure 1. Then, Greek geography can be represented by an object identifier o'' and by associating it with context c , as shown in Figure 1.

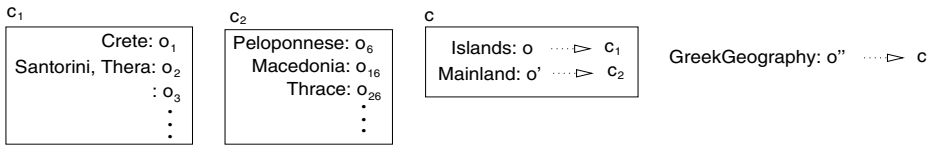


Fig. 1. Bottom-up modeling

The previous examples suggest the following informal definition of context (in its simplest form): a context is a set of objects of interest, each object having zero, one or more names, and zero or one references. Formally, we have the following definition.

Definition 21 Context. A context c is defined as a set of objects, denoted by $objs(c)$, such that each object o is associated with

1. a set of names, called *the names of o in c* , denoted by $names(o, c)$;
2. zero or one context, called *the reference of o in c* , denoted by $ref(o, c)$.

The reason why we use the symbols $names(o, c)$ and $ref(o, c)$, instead of $names(o)$ and $ref(o)$ used in the previous examples, is that an object can belong to different contexts and may have different names and/or reference in each context. That is, *names and references are context-dependent*.

In our previous examples, while explaining the construction of a context, we followed a bottom-up approach. That is, we started from simple objects and built up contexts which were later on referenced by higher level objects (“moving” from left to right in Figure 1). Clearly, we could have followed the opposite construction, i.e., a top-down approach (“moving” from right to left in Figure 1). In fact, as we shall see in other examples, we can also follow a mixed approach.

This flexibility is important in conceptual modeling and implies (among other things) the possibility of *modular design*, i.e., retaining at each level of abstraction the essential information and hiding inessential details (by “encapsulating” them in the form of a context).

Continuing with our examples, let us see a top-down definition of a context. Suppose we are defining a context containing guides to Greece and wish to model a tourist guide as one of its objects. Within this context we represent the guide as follows:

Tourist_Guide: $o_4 \cdots \triangleright c_3$

The next stage is to define the context c_3 that contains the information concerning the tourist guide. The context c_3 is shown in Figure 2.

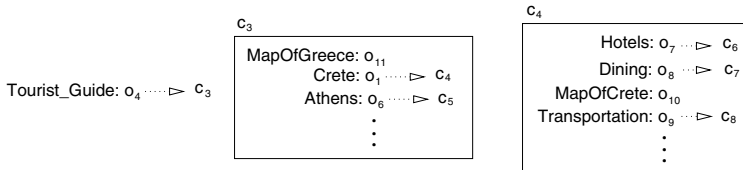


Fig. 2. Top-down modeling

Continuing our top-down design, we now have to define the contexts c_4 , c_5 , and so on. Context c_4 is shown in Figure 2, whereas the remaining contexts are not shown. Context c_4 contains tourist information concerning Crete such as hotels, dining, a map of Crete, transportation, etc. Looking now at the definition of context c_4 , we see that we have to define contexts c_6 , c_7 , c_8 , and so on (their definition is not shown in the figure). Context c_6 will list the hotels in Crete and provide access to further information such as addresses and telephone numbers. Context c_7 will provide access to information about dining in Crete, e.g., a list of restaurants, local dishes, and so on, and context c_8 will give transportation information.

Note that context c_3 shares the object o_1 with context c_1 that we have seen earlier (see Figure 1). Also note that, in context c_3 , object o_1 has a reference (context c_4), whereas in context c_1 , the same object o_1 has no reference.

The notion of context supports a simple and straightforward way of referencing objects at any level of detail. Consider for example the tourist guide of Greece, in Figure 2. Suppose that, currently, we are in the context containing the tourist guide, and we want to look at Cretan hotels. To do so we can "go" from object o_4 (Tourist_Guide) to object o_1 (Crete) and then to object o_7 (Hotels). We indicate this as follows: $o_4.o_1.o_7$, i.e., by forming a path of object identifiers. If the last object in the path has a reference, this points to a context that contains the information of interest.

Several remarks are in order here, concerning our definition of context:

- An object can belong to one or more different contexts. This feature is useful when we want to view an object under different perspectives.
- The same object can have different names in different contexts (in which it belongs). In other words, names are context-dependent. This is very convenient, because a name which has a clearly understood meaning in one context may not do so in another.
- Two different objects can have the same name within a context. As a result, ambiguities may occur when one tries to designate an object of a context using a

name shared with a different object of the same context. We address this problem in [32,33].

- The same object can have different references within different contexts. In other words, references are context-dependent.
- Two different objects, whether or not they belong to the same context, or to different contexts, can have the same reference. This is convenient, as a given context can be reachable through different object paths.
- From within a given context, we can "reach" any object that belongs to the reference of an object within that context (and, recursively, any object that lies on a path).

Roughly speaking, the modeling power of contexts lies in the fact that one can group together quite dissimilar things as the contents of a context, regardless of any structural relationships they may have. In fact, no such relationships are required to hold the contents of the context together.

In summary, modeling with contexts provides several capabilities, including the following: (i) Modeling an object under different perspectives, by associating it with different references in different contexts; (ii) Modular representation, by providing at each level of abstraction an overview of the available information in the form of items that each provide access to relevant detail; (iii) Top-down, bottom-up, or mixed modeling.

Moreover, as we shall shortly see, the combination of contextualization with the traditional abstraction mechanisms provides further modeling capabilities.

3 Structuring the Contents of a Context

Our definition of context allows for some simple relations between objects, for example, two different objects of a context can have the same name and/or the same reference. However, we would like to allow the objects of a context to be related in more complex ways.

For the purposes of this paper, we shall assume that the objects of a context can be structured as in a Telos information base [20].

A Telos information base consists of structured objects built from two kinds of primitive units: *individuals* and *attributes*. An important and distinctive feature of Telos is that individuals and attributes are treated uniformly, and are referred to as "*objects*" (in [20] objects are also referred to as "*propositions*"). Individuals represent entities (atomic ones such as *John*, or collective ones such as *Person*), while attributes represent directed binary relationships between or intrinsic characteristics of entities. Every attribute consists of a *source*, a *label*, and a *destination*. Objects (individuals or attributes) are organized along three dimensions, referred to as the *classification*, *generalization*, and *attribution* dimensions [13,28,20].

Returning now to our notion of context, from now on we shall use the following "enhanced" definition: A context consists of a set of objects, each object having a set of names and zero or one reference (as before), with the following *new* features:

- each object is either an individual or an attribute;
- each object can be related to other objects through instance-of or ISA links;
- each instance-of or ISA link can have zero or one reference.

As a consequence of this enhanced definition of context, each context is assumed to be equipped with (i) a predicate for defining the objects that are attribute links (the remain-

ing objects being individuals), (ii) a predicate for defining instance-of links, and (iii) a predicate for defining ISA links. More precisely, we assume each context to be equipped with three predicates as follows:

1. $attr(obj, from, to)$, declaring that object obj is an attribute link with source object $from$ and destination object to .
2. $in(from, to)$, declaring an instance-of link in which the object $from$ is an instance of the object to .
3. $isa(from, to)$, declaring an ISA link in which the class $from$ is a subclass of class to .

Note that, as attributes are objects, an attribute can have zero, one or more names. Each of these names corresponds to a Telos label.

Consider, for example, modeling employees using a class whose instances have three attributes: name, salary and address. Using our definition of context, this modeling can be done as shown in Figure 3(a), where o is the employee class, and the three attribute declarations define the objects o_1 , o_2 and o_6 as attribute classes from class o to classes o_4 , o_5 and o_3 , respectively. Figure 3(b) shows a more convenient representation of the same context (context c), where the attributes o_1 , o_2 and o_6 are represented by arrows. For example, $attr(o_1, o, o_4)$ is represented by the arrow from o to o_4 . This declares that an instance of the class `Employee` can have an attribute, being instance of class `Name`, that connects it to an instance of class `String`. Note that attribute o_6 has the same name as object o_3 , something allowed by our definition of context. However, if referencing of objects is done through names, this may lead to ambiguities. We address this problem in [32,31]³.

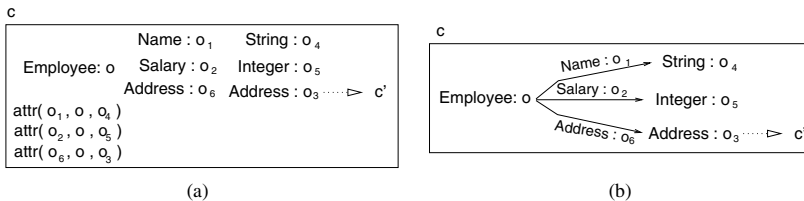


Fig. 3. Modeling an employee using attributes

In the previous example, the employee information is modeled as the contents of a single context, i.e., a context containing the employee class *and* its attributes. An alternative way of modeling an employee is the following: the employee class references a context containing the attribute information. This is shown in Figure 4(a), where the ISA declarations in context c define that `Name` is a subclass of `String` and `Salary` is a subclass of `Integer`. Figure 4(b) shows a more convenient representation of context c , where ISA links are represented by thick arrows. In fact, from now on, we shall use arrows to represent all relationships, with different kinds of arrows for the different relationship types.

³ A simple way to avoid this problem is to put the name of object o_6 in a verb form, e.g., `has_address`.

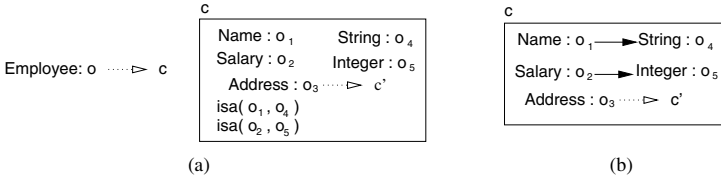


Fig. 4. Modeling an employee using contexts

At first glance, the modeling of Figure 4 looks simpler than that of Figure 3. However, depending on the application, the one or the other could be preferred. The important point here is that the contextualization mechanism offers several alternatives for modeling a given application.

Figure 5 shows another example of context with structured contents, this time using all three abstraction mechanisms, i.e., classification, generalization, and aggregation. It is important to note that contextualization is orthogonal to the other abstraction mechanisms.

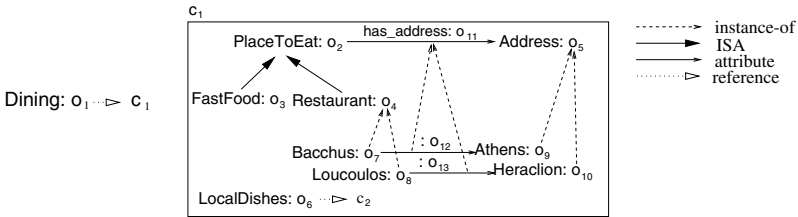


Fig. 5. Structured contents of a context

4 The Interaction between Abstraction Mechanisms

In an information base, the interaction between different abstraction mechanisms provides useful information to the designers of the information base, but also implies certain constraints that must be satisfied to guarantee consistency of the stored information.

For example, if object o is instance of class o' , and o' is subclass of o'' , then o is instance of o'' . This is useful information as it implies that o' inherits all attributes of o'' . On the other hand, if class o_1 is subclass of class o_2 and o_2 is instance of metaclass o_3 , then o_1 cannot be declared as subclass of o_3 , and this is a constraint that must be satisfied to guarantee consistency of the stored information.

The interaction between the traditional abstraction mechanisms of conceptual modeling has been extensively studied in the literature (see [13,28] for a survey). In this section, we study the interaction between the traditional abstraction mechanisms and the mechanism of contextualization.

4.1 Attribution and Contextualization

In our enhanced definition of context, each object can be either an individual or an attribute, and has a set of names and zero or one reference. Now, if the object is an

individual, then its reference can be *any* context. An attribute, however, cannot exist without a source and a destination, and this information must be part of its reference.

Therefore, if the object is an attribute, then its reference must contain at least a description of the source and the destination reference⁴. This can be done as shown in Figure 6. Let o be an attribute from object o_1 to object o_2 with references c_1 and c_2 , respectively. Then, its reference c should contain two special objects: an object o_f named from with reference c_1 and an object o_t named to with reference c_2 .

So from now on, we assume that the reference of every attribute is as shown in Figure 6. That is, it contains information about the source and the destination of the attribute. We also assume that an instance-of or ISA link may have a reference, and that this reference (if any) is of the same kind as the attribute link, i.e., a reference that contains information about the source and the destination of the link.

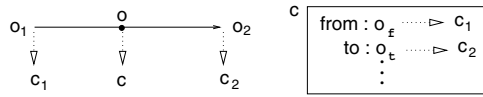


Fig. 6. The reference of an attribute

Of course, apart from the minimal necessary information shown in Figure 6, the reference of an attribute may also contain other information. The question here is whether there are constraints that this "other information" should satisfy.

Let us call *traversal path* any path from an object in the source reference of the attribute to an object in the destination reference, such that every member of the path is an attribute, instance-of, or ISA link. We call *attribute path* any traversal path, every member of which is an attribute. Intuitively, an attribute path defines an attribute from an object in the source reference to an object in the destination reference.

Now, the constraint that we propose for the reference of an attribute can be stated informally as follows: the attribute must collectively represent all traversal paths from objects in its source reference to objects in its destination reference. Clearly, in order this requirement to be satisfied, all traversal paths must be attribute paths. Hence the following constraint on the information that the reference of an attribute can contain:

Constraint 41 Attribute Reference Constraint. Every traversal path in the reference of an attribute is an attribute path.

Figure 7 illustrates the interaction between attribution and contextualization in a top-down modeling of demographic data. The reference of attribute o_4 (Related_To) is context c_4 . This reference contains two traversal paths that are both attribute paths. The first of these paths goes from object o_5 in context c_2 to object o_8 in context c_3 , and consists of a single attribute: o_{11} (born_in). Within context c_4 , this is defined as $attr(o_{11}, o_f.o_5, o_t.o_8)$ because objects o_f and o_t refer to the source and destination references of attribute o_4 , respectively. The second path goes from object o_7 in context c_2

⁴ We refer to the reference of the source (resp. destination) of an attribute as the *source* (resp. *destination*) reference.

to object o_8 in context c_3 and consists of two attributes: o_{12} (*works_for*), from o_7 to o_{13} , and o_{14} (*located_in*), from o_{13} to o_8 . Note that in Figure 7(a), context c_4 is given in a pictorial way, where the special objects o_f and o_t are omitted and predicates are depicted through arrows. Its actual definition is given in Figure 7(b).

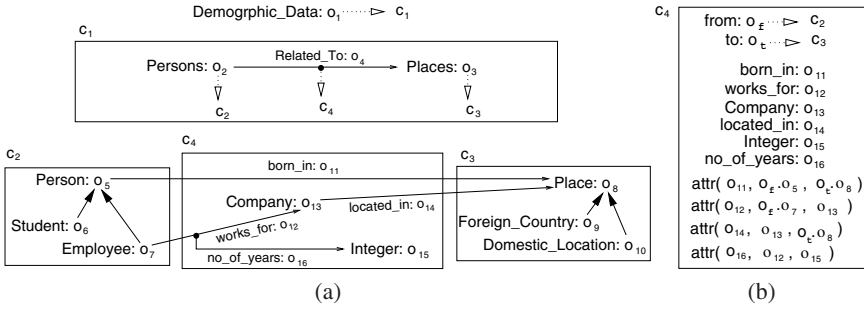


Fig. 7. Top-down modeling of demographic data

4.2 Classification and Contextualization

The interaction between classification and contextualization is similar to that between attribution and contextualization. Thus the reference of an instance-of link l should contain only instance-of links from objects in the source reference of l to objects in the destination reference of l .

Constraint 42 Instance-of Reference Constraint. Every traversal path in the reference of an instance-of link consists of a single instance-of link.

If an object o is an instance of object o' then the reference of the instance-of link may classify objects in the reference of o into object classes in the reference of o' . Intuitively, we can say that the objects in the reference of o follow the "schema" defined in the reference of o' .

Probably the most relevant example of this interaction is the one relating a database schema with its instances. In Figure 8(a), object o (Instance) is instance-of object o' (Schema). Note that the reference c_{in} of the instance-of link contains only instance-of links from objects of c to objects of c' .

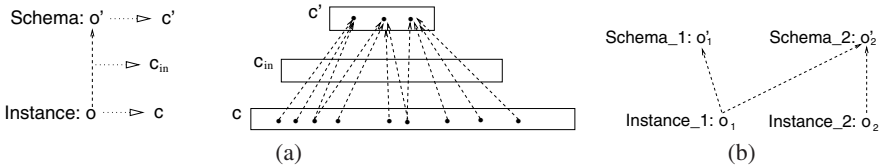


Fig. 8. The reference of an instance-of link

Intuitively, within c_{in} , objects of c are instances of objects of c' . For example, if c contains a set of relational tuples and c' contains a relational database schema then the

instance-of links relate tuples in c with tables in c' . That is, the contents of c_{in} can be seen as a sort of metadata describing the association of instances to schemas.

Note that the separation between instance and schema allows for several sets of objects to share the same schema, and the same set of objects to be classified under different schemas. For example, in Figure 8(b), schemas o'_1 (Schema_1) and o'_2 (Schema_2) share the same instance set, represented by object o_1 (Instance_1). On the other hand, instance sets o_1 and o_2 share the same schema o'_2 (Schema_2).

4.3 Generalization and Contextualization

Generalization establish a subclass-superclass relation between classes and it is used to emphasize the similarities among classes with common superclasses and to hide their differences. The interaction between generalization and attribution is expressed by the well known mechanism of *attribute inheritance*. In our framework, in addition to attribute inheritance, we support a new mechanism called *reference inheritance* related to the interaction between generalization and contextualization. Roughly speaking, according to reference inheritance, the reference of the subclass *inherits* the contents of the reference of the superclass.

Formally, reference inheritance is defined through a partial order over contexts that we call *context refinement*.

Definition 41 Context refinement. We say that context c *refines* context c' , or that context c is a refinement of c' , iff (i) every object of c' is also an object of c , (ii) the names of every object in c' are included in the names of the object in c , (iii) every predicate of c' is also a predicate of c , and (iv) the reference of every object of c' refines the reference of the object in c .

Note that the above definition of context refinement is recursive and that every context is a refinement of itself. We show in [30] that refinement is a partial pre-ordering, i.e., reflexive and transitive. Moreover, we show that context refinement is a partial ordering up to context equivalence, where context equivalence is defined as follows: two contexts are *equivalent* if they have (i) the same objects, (ii) the same names for each object, (iii) the same predicates, and (iv) the references of each object in the two contexts either both do not exist, or both exist and they are equivalent. Roughly speaking, two contexts are equivalent if they have the same contents, up to equivalence of the object references.⁵

The following constraint expresses the application of reference inheritance on ISA links.

Constraint 43 Reference Inheritance Constraint. The source reference of an ISA link refines the destination reference of the link.

For example, in Figure 9, object o_2 (Hospital) is a subclass of object o_1 (Organization). The source reference of this ISA link (context c_2) is a refinement

⁵ Notice the similarity between context equivalence and deep object equality in object oriented databases [2].

of the destination of the link (context c_1), as c_2 contains all the contents of c_1 . Therefore, the reference inheritance constraint is satisfied. Intuitively, we can say that the contents of c_1 have been inherited by c_2 .

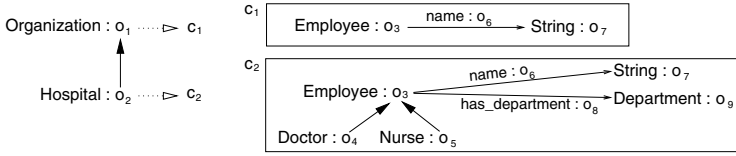


Fig. 9. Interaction between generalization and contextualization

To keep the contexts concise, we could eliminate duplications in the contents of the source reference of the ISA link. In this case, the complete contexts are obtained after the application of reference inheritance on the ISA links. This, however, is an implementation issue that lies beyond the scope of this paper. A mechanism for eliminating duplications is proposed in [30].

Context refinement can be achieved in stages through the repetitive application of the following operations on the contents of a context: (i) the addition of a new object (possibly an attribute), (ii) the specialization, generalization, or classification of an object (possibly an attribute), (iii) the addition of a name to an object, (iv) the addition of a reference to an object or link, and (v) the application of the previous operations to the contents of a reference. The resulting context is certainly a refinement of the original context, as it merely extends the information contained in that context (and no cancellation takes place).

5 Related Work

Various forms of contextualization have appeared in the area of computer science. However, these forms are very diverse and serve different purposes. We choose for comparison with our work the more closely related approaches which we classify in two topics: general contextualization frameworks and semantic model clustering.

5.1 General Contextualization Frameworks

In this subsection, we compare the present work with our own previous works [32,33], as well as with the work of Mylopoulos and Motschnig-Pitrik [21,22]. These works attempt to introduce a general framework for contextualization in information bases.

In [33], a naming mechanism based on the concept of context is proposed, in order to resolve naming problems that arise in information bases structured with the traditional abstraction mechanisms of classification, generalization, and attribution. A context is identified by a node and a link class, called *pivot elements* of the context. The contents of a context consist of objects and links which are associated with the pivot elements of the context. This definition of context allows the information base to be decomposed into partitions on the basis of one of the traditional abstraction mechanisms. Finally, relative naming is supported, as well as nesting of non-overlapping contexts.

In [21,32], a context is treated as a special object which is associated to a set of objects and a lexicon, i.e., a binding of names to these objects. These works support nesting of contexts, context overlapping, relative naming, and define operations on contexts, such as context union, intersection, and difference. In addition, [32] establishes properties of operations on contexts, such as commutativity, associativity and distributivity. On the other hand, [21] considers issues such as authorization and transaction execution.

The notion of context introduced in this work, still supports nesting of contexts, context overlapping, and relative naming, as described in our previous works [31,32], yet it advances with respect to [21,32,33] mainly along two lines:

- We distinguish between objects and contexts. Objects represent real world concepts, whereas contexts are collections of objects. Within each context, local names and semantics are assigned to objects, as well as references (which are also contexts) for describing objects in more detail. Thus, a real world concept (e.g., the geographical viewpoint of Greece) is represented by an object which can have different detailed descriptions within different contexts (e.g., the 15th century and the 20th century). This is certainly a modeling capability not offered by the other approaches.
- We support the interaction of our contextualization mechanism with the traditional abstraction mechanisms. Works in [21,22,32], [33] lack this interaction. In [22], the notion of context is introduced in the Telos data model, where each context is considered to be at the Token level, i.e., an atomic object, and contexts do not participate in classification or generalization hierarchies.

5.2 Semantic Model Clustering

In "real life" applications, it is often the case that semantic data models become large and complex, and thus difficult to understand. Several techniques cope with this problem by decomposing the global schema into smaller, more manageable partitions, called *entity clusters* [7,29,5,34], [4,8], [36,35].

In [29], several kinds of clustering are defined, all of which are supported by our framework:

1. *Dominance grouping*: Here, an object o is grouped together with its related objects into a cluster that represents the same real world entity as o , but at a different level of abstraction. In our framework, we support this kind of grouping by allowing the reference of an object to contain the object itself.
2. *Abstraction grouping*: Here, objects participating in abstractions such as classification, generalization, and attribution are grouped in a cluster. In our framework, we support this kind of grouping by allowing objects related by instance-of, ISA, and attribute links to be grouped together with these links in a context.
3. *Relationship grouping*: Here, a relationship together with its participating entities are grouped into a cluster. In our framework, we support this kind of grouping, as relationships are represented by attributes and a context may contain any kind of object, i.e., individual or attribute.

In all approaches in the literature, dominance grouping is based on the following name convention: each cluster should have the same name as the object it represents.

By contrast, in our framework, dominance grouping is based on object ids, allowing objects to have different names at different levels of abstraction.

In [5], an additional kind of clustering is defined, called *relationship abstraction*, which abstracts a number of relationships into a higher-level relationship. In our framework, we support this kind of clustering through the concept of reference.

Our framework differs substantially, from all of the above works in the following:

- A global schema is not a requirement for modeling the real world. Rather, it is possible that information about an object can only be found scattered across contexts.
- We support relative naming and relative semantics w.r.t. a context. Within different contexts, information about the same object can even be conflicting. Thus, information is meaningful only within a context, and its validity outside it cannot directly be assumed (unless explicitly declared).
- We distinguish between objects and contexts, with the advantages described in the previous subsection.
- We support the interaction of our contextualization mechanism with the traditional abstraction mechanisms.

6 Conclusions

In this paper, we are concerned with a notion of context in the area of conceptual modeling. In our approach, a context is seen as a structured set of objects, in which each object is associated with a set of names and (possibly) a reference: the reference of the object is another context which "hides" detailed information about the object. Within a context, objects can be structured through the traditional abstraction mechanisms of classification, generalization, and attribution. One of the contribution of this paper is that we study how the contextualization mechanism interact with the traditional abstraction mechanisms.

Currently, we are working on the development of a general framework for querying information bases which support contextualization. This framework includes the definition of useful fundamental query operations on contexts such as selection, projection, and join, as well as composition operations such as union, intersection, and difference.

Acknowledgements

We would like to thank Aran Lunzer for his comments.

References

1. S. Abiteboul and A. Bonner. Objects and Views. In *Proc. ACM-SIGMOD Conference*, pages 238–247, Feb. 1991. 475, 475
2. S. Abiteboul and J. Van Den Bussche. Deep Equality Revisited. In T. Ling, A. Mendelzon, and L. Vieille, editors, *Proc. of DOOD'95*, volume 1013, pages 213–228, Dec. 1995. 485
3. F. Bancilhon and N. Spyrtatos. Update Semantics of Relational Views. *ACM Trans. Database Syst.*, 6(4):557–575, Dec. 1981. 475, 475
4. L. Campbell, T. Halping, and H. Proper. Conceptual Schemas with Abstractions: Making Flat Conceptual Schemas More Comprehensible. *DKE*, 20(1):39–85, June 1996. 475, 476, 487

5. B. Czejdo and D. Embley. View Specification and Manipulation for a Semantic Data Model. *IS*, 16(6):585–612, 1991. 475, 476, 487, 488
6. N. Delisle and M. Schwartz. Contexts: A Partitioning Concept for Hypertext. *ACM Trans. Office Inf. Syst.*, 5(2):168–186, Apr. 1987. 475
7. P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *Computer J.*, 29(4):348–360, Apr. 1986. 475, 476, 487
8. M. Gandhi, E. Robertson, and D. Gucht. Levelled Entity Relationship Model. In *Proc. of ER'94*, pages 420–436, Manchester, U.K., Dec. 1994. Springer-Verlag. 487
9. G. Gottlob, P. Paolini, and R. Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, Dec. 1988. 475, 475
10. G. Gottlob, M. Schrefl, and B. Röck. Extending Object - Oriented Systems with Roles. *ACM Trans. Inf. Syst.*, 14(3):268–296, July 1996. 475, 475
11. G. Hendrix. Encoding Knowledge in Partitioned Networks. In N. Findler, editor, *Associative Networks*. New York: Academic Press, 1979. 475
12. G. Hendrix. Encoding Knowledge in Partitioned Networks, 1979. In [?]. 475
13. R. Hull and R. King. Semantic Database Modeling. *ACM Comput. Surv.*, 19(3):202–260, Sept. 1987. 476, 480, 482
14. V. Kashyap and A. Sheth. Semantic and Schematic Similarities between Database Objects: A Context-Based Approach. *Vldb Journal*, 5(4):276–304, Dec. 1996. 475, 475
15. R. Katz. Towards a Unified Framework for Version Modeling in Engineering Databases. *ACM Comput. Surv.*, 22(4):375–408, Dec. 1990. 475, 475
16. S. Matwin and M. Kubat. The role of Context in Concept Learning. In *Proc. of the ICML-96, Workshop on Learning in Context-Sensitive Domains*, pages 1–5, Bari, Italy, July 1996. 475, 475
17. J. McCarthy. Notes on Formalizing Context. In *Proc. IJCAI-93*, pages 555–560, Chambéry, France, 1993. 475, 475
18. R. Michalski. How to Learn Impressive Concepts: A Method Employing a Two-Tiered Knowledge Representation for Learning. In *Proceedings of the 4th International Workshop in Machine Learning*, pages 50–58, Irvine, CA, 1987. 475, 475
19. R. Motschnig-Pitrik. An Integrated View on the Viewing Abstraction: Contexts and Perspectives in Software Development, AI, and Databases. *Journal of Systems Integration*, 5(1):23–60, Apr. 1995. 475
20. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM Trans. Inf. Syst.*, 8(4), Oct. 1990. University of Toronto. 480, 480, 480
21. J. Mylopoulos and R. Motschnig-Pitrik. Partitioning Information Bases with Contexts. In *Proc. of CoopIS'95*, pages 44–55, Vienna, Austria, 1995. 475, 476, 486, 487, 487, 487, 487
22. J. Mylopoulos and R. Motschnig-Pitrik. Semantics, Features, and Applications of the View-point Abstraction. In *Proc. of CAiSE'96*, pages 514–539, Heraklion, Greece, May 1996. 476, 486, 487, 487
23. A. Ouksel and C. Naiman. Coordinating Context Building in Heterogeneous Information Systems. *J. of Intelligent Inf. Systems*, 3(2):151–183, 1994. 475
24. J. Richardson and P. Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proc. ACM-SIGMOD Conference*, pages 298–307, Denver, Colorado, May 1991. 475, 475
25. M. Scholl, C. Laasch, and M. Tresch. Updatable Views in Object-Oriented Databases. In *Proc. of DOOD'91*, pages 189–207, Munich, Dec. 1991. 475
26. E. Sciore. Object Specialization. *ACM Trans. Inf. Syst.*, 7(2):103–122, Apr. 1989. 475, 475
27. J. Shilling and P. Sweeney. Three Steps to Views: Extending the Object-Oriented Paradigm. In *Proc. OO Prog., Syst., Lang. and Appl. - OOPSLA*, pages 353–361, Oct. 1989. 475

28. V. Storey. Understanding Semantic Relationships. *VLDB Journal*, 2(4):455–488, Oct. 1993. 480, 482
29. T. Teorey, G. Wei, D. Bolton, and J. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Commun. ACM*, 32(8):975–987, Aug. 1989. 475, 476, 487, 487
30. M. Theodorakis. *Contextualization: An Abstraction Mechanism for Information Modeling*. PhD thesis, Department of Computer Science, University of Crete, Greece, 1998. 485, 486
31. M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. A Theory of Contexts in Information Bases. Technical Report 216, ICS.FORTH - Hellas, Mar. 1998. 475, 481, 487
32. M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. Context in Information Bases. In *CoopIS'98*, pages 260–270, NY City, USA, Aug. 1998. IEEE Computer Society. 476, 476, 480, 481, 486, 487, 487, 487, 487, 487
33. M. Theodorakis and P. Constantopoulos. Context-Based Naming in Information Bases. *International Journal of Cooperative Information Systems*, 6(3 & 4):269–292, 1997. 475, 476, 480, 486, 486, 487, 487
34. O. Troyer and R. Janssen. On Modularity for Conceptual Data Models and the Consequences for Subtyping, Inheritance & Overriding. In *Proc. of Ninth Int. Conf. on Data Eng.*, pages 678–685, Vienna, Austria, Apr. 1993. IEEE Computer Society. 487
35. D. Vermeir. Semantic Hierarchies and Abstractions in Conceptual Schemata. *IS*, 8(2):117–124, 1983. 487
36. H. Weber. Modularity in Data Base System Design: A Software Engineering View of Data Base Systems. *VLDB Surveys*, pages 65–91, 1978. 475, 476, 487

Architectures for Evaluating the Quality of Information Models - a Meta and an Object Level Comparison

Reinhard Schuette

Department of Production and Industrial Information Management,
University of Essen, UniversitaetsstraÙe 9, D-45141 Essen
E-Mail: reinhard.schuette@pim.uni-essen.de

Abstract. The evaluation of information models is an outstanding research field in information systems engineering. From a theoretical oriented research viewpoint, it has to be questioned whether it is possible to evaluate artefacts with respect to philosophical and decision theory oriented aspects. Basing on the theoretical assumption that information modeling is a decision problem, this article deals with model evaluation approaches discussed in literature. Approaches will be compared not only on a meta level (ontological and epistemological assumptions) but also on an object level (evaluation of information models). The Frameworks of Moody/Shanks, Krogstie/Lindland/Sindre and the Guidelines of Modeling will be described and compared.

1 Motivation for model evaluation

Information models can be defined as the representation of a system which has been constructed by subjects and have been made explicable with the means of language. One of the main issues within the field of prescriptive modeling theory deals with the problem of efficiently selecting modeling measures which lead to suitable models, taking into account the basic modeling conditions.

The analogies between software engineering and the construction theory in engineering science were repeatedly pointed out in the past. They both have in common that the early development stages of either new products (construction phase) or new application systems (requirements engineering) are of great significance. However, the construction of material products can be compared to the design of immaterial products only to a very limited extent. On closer examining the information modeling process, it becomes obvious that the product to be designed, the model, highly depends on the designer and the design process. As a consequence, the evaluation criteria to be applied differ from the criteria valid for material production.

2 Evaluation framework set-up

First of all, the meaning of the term “quality” has to be defined. Regarding the diversity and complexity of alternative quality-definition approaches, efforts for systemizing this term do not lead to any successful results. The following part of this paper is

based on the assumption that “quality” means fulfilling requirements. In contrast to former opinions, a technocratic quality perspective is not longer sufficient for evaluating the quality of a performance. This understanding of quality is not only valid for the production of goods. Being applied as a general quality-evaluation philosophy for each type of company activities, it is also applied in the field of software development. The quality of information models is based on an analog quality understanding, i.e. in how far the requirements of different model-addressees are fulfilled.

Bearing in mind the requirements-oriented understanding of quality, this document presents a comparison of alternative concepts for evaluating information models including a relative and, at the same time, concept-exogenous comparison of the models’ advantages. *At first*, comparison will be made on a meta level, describing the weltanschauung the approaches are based on. (see section 2.1). *The second* step on object level includes the examination on the problem of quality evaluation handling in the different approaches, taking into account the general reflections on decision-theory (see section 2.2).

2.1 The significance of Weltanschauung in information systems modeling

The term Weltanschauung (“world view”) stands for the way the designer sees the world [Inwo95]. The differentiation between realism and idealism on the one hand and the ontological and epistemological position on the other leads to three different world views. The ontological realism is based on the assumption that the existing world is independent from us whereas the ontological idealism (nominalism in social science [BuMo79]) denies that basic position. According to the epistemological realism, objective knowledge is possible as we recognize the world as it is. Within epistemological idealism, however, the source of all knowledge is the spirit. As a consequence, we cannot recognize the world as it is without the existence of a knowing spirit. Keeping to the differentiation between ontology and epistemology and not going into a more detailed differentiation (e.g. Resc97), the scientist can take up three optional “knowledge theories”. A combination of ontological and epistemological realism is practicable (e.g. naive realism, critical realism), but not stringent. Ontological idealism can only be combined with the epistemological idealism as the non-existing reality cannot be objectively perceived (This is valid especially for the radical constructivism). The combination of ontological realism and epistemological idealism is placed right in the middle of realism and idealism.

The different world views of the individual designers, which serve as the basis for the different model-quality evaluation approaches cannot be evaluated because of the paradigmatic incommensurability problem connected. Due to the impossibility of making metaphysical assumptions subject to “unconditioned” criticism, the only real fact to be examined is the coherence between model-evaluation approaches and the assumptions on the basic philosophy of the designer. For avoiding architectures with incoherent messages, the importance of a meta level for defining criteria, scales, etc. on an object level has to be taken into account. Furthermore, the assumptions underlying the different approaches can be inferred from the different world views, e.g. a person supporting a critical-realistic philosophy is likely to chose an evaluation approach different from the more subjective approach chosen by an epistemologically-oriented designer.

2.2 Decision-theory oriented observations about model evaluation

From the decision-theory point of view, the following components for model evaluation, which sketch the rudimentary decision-model part, have to be taken into consideration. [Bitz81, Laux98]:

- First of all, the *action-variables* of information modeling have to be identified. These variables can be seen as measures taken by the designers for achieving their model-construction objectives. Out of a vast amount of modeling measures, the designer selects those measures best-suited for completing his work. The selection of alternative measures is closely related to the type of results the designer wants to produce, especially when considering all the information-economic aspects. The knowledge about the intended objectives serves as the basis for deciding which alternative measures can really be integrated into the decision-model.
- Together with the states which, after application of a prediction function, can be considered realistic, the measures will be transferred into a *result-matrix*. In case of different real-life situations, alternative modeling measures can lead to different results (consequences). For determination of results the designer has to take into account that the results may have differing scale-levels. As a consequence, various methods for prognosis and measurement are required. For examining modeling measures over a period of time, prognoses are required.
- After the application of a modeling measure, the results will be evaluated on the basis of the subjective preferences of the model-addressee (preference function) and will be converted into utility values reflecting the preferences of the designer (decision-maker). In case of differing preferences of model designer and model addressee, the designer has to predict the model addressee's preference function. Through application of a preference-function, the result vectors of an alternative action (measure) will be condensed to a preference value which reflects the level of advantage of the action-variable. The preference function in this case has to take into account the relevant results (action consequences). According to a gradation consistently supported in literature, the preference function is composed of four elements: the preference of type, degree, safety, and time. When applying a specific *preference of degree*, the decision-maker makes clear which preference he assigns to a result according to its degree of markedness. A differentiation is normally made between the extreme, satisfaction and improvement [Bitz81]. The *Preference of type* is another result feature to be evaluated whenever the decision maker is trying to achieve several goals which are neither neutral nor complementary. Through application of the preference of type, the various goals can be weighted. *Preference of time* is utilized whenever the results cannot be realized at the same time. The *Uncertainty preference* is used for evaluating the uncertainties of states connected to the consequences of an action. Whenever the states can be exactly defined, the uncertain consequences will be evaluated with the help of the preference values.

The decision model components can be found in figure 1.

3.1.2 Components of the approach

The intention of MOODY/SHANKS is to evaluate models for individual criteria with the help of metrics so that, after having weighted the results, the evaluation can be made on model level [MoSh94]. On the basis of the data model objectives, followed by the evaluation of alternative strategies, the best-suited alternative actions will be derived, resulting in improved model quality.

The approach is composed of the following individual components (see also figure 2):

- A model is defined as a linguistic representation interpreted and designed by stakeholders.
- The evaluation framework is composed of eight *quality factors*: completeness, integrity, flexibility, understandability, correctness, simplicity, integration and implementability. *Completeness* is achieved when all information needed by the user is included in the model. *Integrity*, as part of the evaluation framework, is achieved when the “business rules” which have to be realized by the data are also completely covered by the data model. By this means, the interaction with the dynamic view in the data model is pointed out. *Flexibility* is concentrating on those features of the model which are needed for future low-effort model adjustments. *Understandability* of a model is achieved when the model contents can be easily understood by the model addressee so that no additional explanations are required. *Correctness* means following the rules of the modeling-technique (e.g. information objects have to be named, entity types have to be allocated to primary keys) (syntactical or grammatical correctness). Correctness also means a presentation free of redundancies in the form of an internal lack of redundancy (e.g. the definitions of two entity types should not overlap). *Simplicity* of a model is required as it is needed for designing a model which is more flexible, easy-to-implement and easy understandable [MoSh94]. The *Integration* aspect is focusing on the consistency between the data model to be designed and the data already existing in the company. For evaluating not only the technical aspects, but also those of time and costs, the *Implementability* aims at keeping the implementation risks low. *Quality Metrics* help measuring the quality factors. The syntactic quality of a model, for example, will be determined through the number of syntactic mistakes.
- Those persons interested in designing or using models are called *Stakeholder*
- Strategies for fulfilling the requirements of the quality factors are called *Improvement Strategies*.
- The process of *Weighting* allocates evaluation weights to the quality factors for representing the relative significance of a quality factor. The weighting of an evaluated quality factor and the addition of all these quality factors allows the calculation of a weighted quality characteristic for a model. This characteristic serves for calculating the order of rank.
- Contrary to the weighting of quality factors, the rating determines the value of a quality factor in one individual model.
- For considering also the process quality, Review Checkpoints are taken into account [MoSD98], which, at certain points of time, are used for examining the quality factors with the help of the stakeholders.

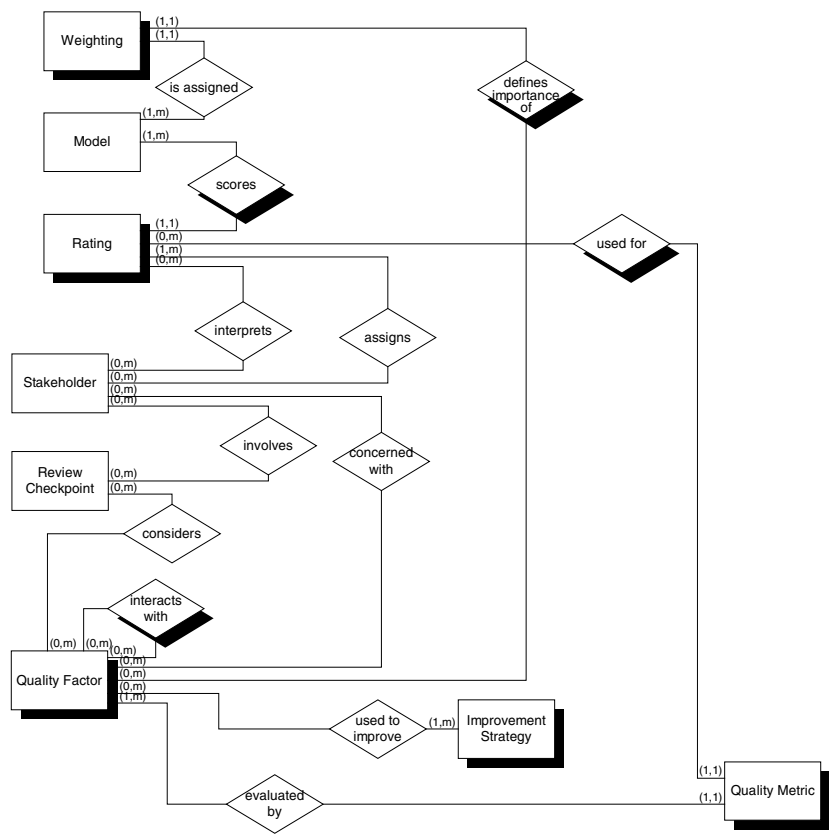


Fig. 2. Meta model of the approach by Moody/Shanks [ShDa97, Mood98]

3.2 The approach by KROGSTIE/LINDLAND/SINDRE

The approach made by KROGSTIE ET AL., which represents an expansion of the framework originally introduced by LINDLAND ET AL., is following a communications structure which consists of six semiotic levels. It is mainly focusing on the framework of [LiSS94] and was further developed by KROGSTIE in his thesis. [Krog95]. Afterwards the development of the original framework was also supported by LINDLAND and SINDRE [KrLS95a, KrLS95b].

3.2.1 Weltanschauung of the approach

Similar to the results of the FRISCO-project [Falk97] the approach follows a view which is oriented towards a socially-constructed reality, combining an ontological realism with epistemological idealism. Consequently, the approach follows an epistemological position which is oriented towards subjectivity.

3.2.2 Components of the approach

The approach by KROGSTIE ET AL. Is composed of the following components (see figure 3)²:

- The *Audience*, consisting of the various Participants with the participants representing a special type of Stakeholders.
- The *Modeling Domain*, which is independent from the modeling subjects [Krog95, p. 97]. When considering the time factor, various domain types (Domain Type, e.g. existing information system, future information system) can be differentiated.
- The *Language* necessary for model design.
- The *Participant Knowledge* about the domain to be modeled, which is needed for model design.
- The *Model Interpretation* done by a participant.

The framework as defined by KROGSTIE ET AL is mainly concentrating on the type of model evaluation. On the semiotics level, a distinction between technically-oriented aspects (physical, empirical, syntactical) and social communication aspects (semantic, pragmatic and social) was made. The framework is focusing on the three main criteria which follow semiotics: Syntax, Semantics and Pragmatics.

The *Syntactical Quality* of a model is given when the number of morphological errors (model-completeness compared to the meta model) is low and when the model is syntactically complete (consistency of the model as compared to the meta model).

The *Semantic Quality* depends on the relationship between domain and model. When all messages within the model are also relevant for the domain and when these messages are representing exactly the facts to be found in the domain, then the requirements of model validity is fulfilled. A model can be called complete if all messages relevant for the domain are also enclosed in the model. The validity emphasizes the quality of the model's messages regarding reality whereas completeness is achieved when all aspects found in reality have been completely transferred into the model. Due to the lack of resources, the criteria for semantic quality will be qualified by turning them into feasible criteria. The objectives of a model are therefore called feasible validity and feasible completeness. The *perceived semantic quality* represents the agreement of all participants of the model interpretation with their knowledge about the domain.

The *pragmatic quality* of a model will also be cut-down through application of the feasibility concept. A model can be called easily comprehensible when all model-addressees are in a position to understand the meaning of the model. No closer examination was made on the language and social quality.

² Due to clarity reasons, the quality factors in the meta model were shown in a redundant way.

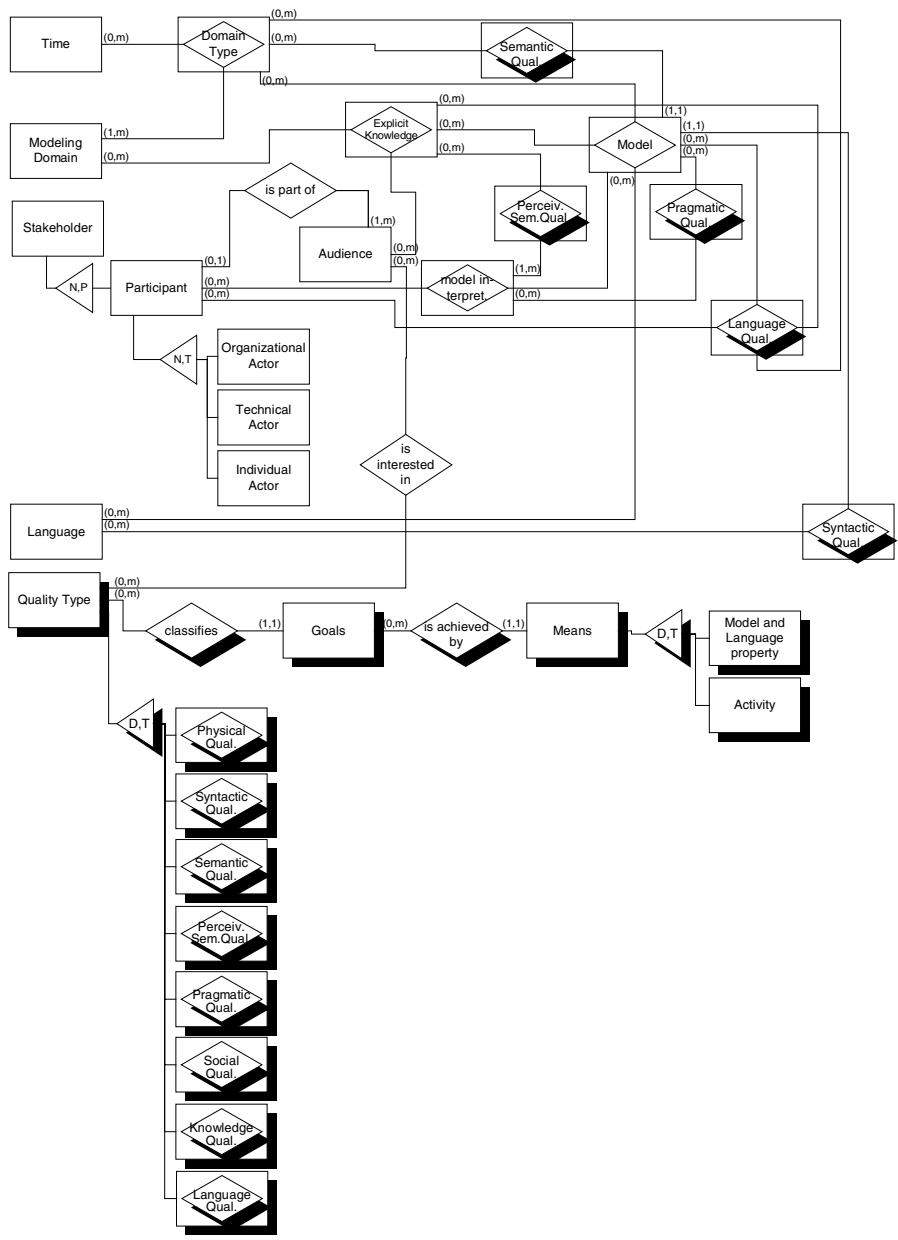


Fig. 3. Meta model of the approach by KROGSTIE/LINDLAND/SINDRE

3.3 The guidelines of modeling

The Guidelines of Modeling (GoM) represent an approach for evaluating the quality of a model. The idea of the GoM was constructed in analogy to the Generally Accepted Accounting Principles (GAAP). A detailed description of the GoM can be found in [Schü98, ScRo98]. The process of defining modeling conventions is described in [Schü98].

3.3.1 Weltanschauung of the approach

The approach is based on a moderate constructivistic position. The ontological realism is supported as well as the epistemological idealism. Although direct access to an original which is independent of human beings is denied, the existence of the original itself is not being denied. The subjectivity of perceiving and the faculty of cognition are highly estimated and trivial conclusions from the appearance to the reality are denied.

3.3.2 Components of the approach

The meta model for the Guidelines of Modeling consists of the following components:

- All information modeling efforts focus on the *subject*.
- Not only is the modeling *language* a means for model-representation. Language is also dominating a subject's type of conceptualization. As a consequence, the utilization of specific language also affects the model domain. "The structure of the world's facts or mere optional facts is not independent of language." [Steg70, p. 15].
- Besides the already designed problem domain, the object to be evaluated, the *information model*, is also determined by the modeling language and the system aspect.
- Information models are always designed for a special purpose. Without having a special purpose in mind, a model could not be structured in orientation to a specific goal.
- Besides the aspects which highlight the model construction, it has to be pointed out that, according to the GoM, the evaluation of models is following the assumptions as pointed out in the basic decision-theory model. There are some result types which serve for model evaluation. The *general principles* of modeling can be understood as abstract objective to be achieved.
- Extensions are being made to the result types potentially provided by the GoM approach. These extensions are made by subjects and include the *preference functions* consisting of the four preference types: Result, degree, time and safety.
- The preference function is the decisive factor for further evaluating individual *conventions* (Measures) for enhancing the quality of a model. A differentiation is made between *view-specific* and *language-specific* conventions. View-specific conventions contain the description of conventions for the individual system-theory oriented viewpoints. On the one hand, they are language-dependent as evaluation without having a rudimentary knowledge of language in mind. On the other hand, these conventions can be formulated on a "reference-language" level being independent of an individual language. When modeling the dynamic system

structure, for example, a convention can be applied for naming the active elements as the process-oriented modeling language normally does not have such a construct. Conventions can, however, only be properly evaluated within a concrete context. Result types and their aggregations include the construction adequacy, language adequacy, economic efficiency, clarity, comparability and systematic design. For evaluating the appropriateness of a model construction, the *Principle of construction* adequacy is applied. On the one hand, the construction quality of a model is being achieved when – from the model addressee's standpoint – a consensus can be found according the presented *problem*. On the other hand, a consensus about the type of model construction has to be defined, i.e. a consensus about the *model representation* needs to be settled. Whereas the principle of construction adequacy focuses on the evaluation of the problem representation in the model, the *principle of language adequacy* focuses on the interrelation between the model system and the utilized language. This includes the differentiation of language *suitability* and language *correctness*. An economic restriction is formulated through the *Principle of economic efficiency*. The *Principle of clarity* deals with the comprehensibility and explicitness of a model. The objectives of clarity include an addressee-oriented hierarchical decomposition, layout design and filtering of information. The *Principle of systematic design* focuses on the generally accepted differentiation between the structure and the behavior of information systems, demanding inter-model consistency between structural and behavioral models. Two models are being compared on a semantic level with the *Principle of comparability*, i.e. the content of two models shall be compared regarding correspondence and similarity.

- Without going into details about the goal(s) to be achieved in modeling [see Schü98], we want to take a closer look in the interrelation between conventions and the different specification levels of the GoM. The goals to be achieved by utilization of the general principles are similar to the quality factors of the other approaches. When being assigned to the different objective-levels, these goals can be evaluated within a specific context. We make no differentiation between quality factors and the goals of modeling as we support the assumption that quality is the generic target which corresponds to the business management objectives.

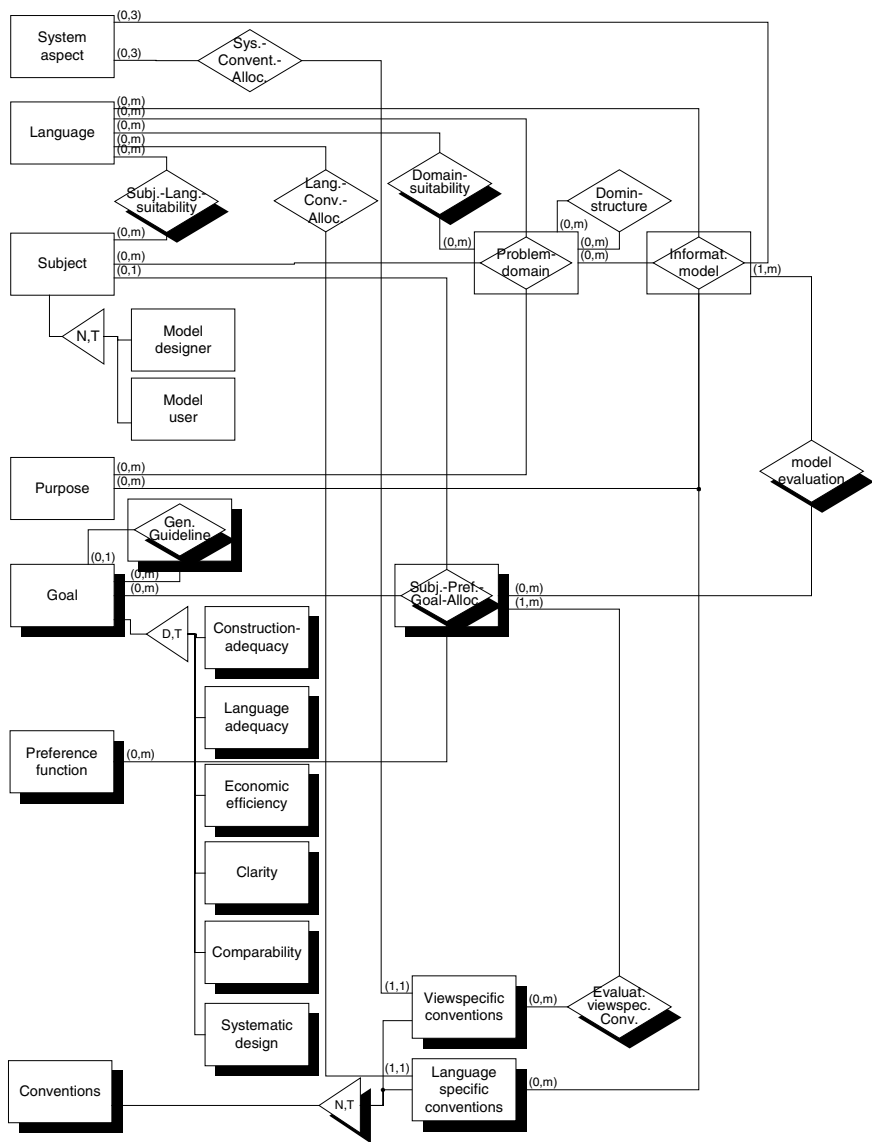


Fig. 4. Meta model of the GoM

4 Comparison and evaluation of architectures

4.1 Meta level: Comparison of the Weltanschauungen of the approaches

The architectures outlined in the previous sections are all based on an ontological realism. The epistemological positions follows idealistic viewpoints (subjectivity, constructivism).

The individual world view of a subject was not thoroughly taken into account in the approach made by MOODY/SHANKS. With the following examples, the problem of transferring the basic world views to the object level is made explicit:

- A domain is implicitly considered a part of reality which is existing independently of the modeling subject (designer). From a critical realist's perspective, this viewpoint is indeed a consequent one. However, from an epistemological subjectivism's perspective, this is not the case, as the domain is understood as an entity already designed by subjects.
- When formulating the quality factors, every once in a while a demand for completeness and relevance regarding the real system is expressed. [ShDa98]. According to these formulations, the question remains open whether these demands correspond to the epistemological position. According to [Mood98] completeness is related to the requirements of the model addressee, so that the danger of inconsistency is avoided as the quality factors are only checked against the conceptualization of the user. Whenever checking the quality factors directly against reality, though, (checking completeness by means of ontological characteristics) this would result in a violation of the basic position as subjectivistic positions allow no direct access to reality.

KROGSTIE ET AL. [KrLS95a] follow a constructivistic evaluation assumption. Their position is based on an ontological realism which is related to a constructivistic epistemological position. However, the demand for higher integration of pragmatic consideration and the actual standpoints of the authors' seem to conflict. An example for this can be found in [Krog95]:

- "The primary goal for semantic quality is a *correspondence* between the externalized model and the domain as before, but this correspondence can neither be established nor checked directly: to build the model, one has to go through the 'participants' knowledge regarding the domain, and to check the model one has to compare this with the participants' interpretation of the externalized model." [Krog95, p. 99]. These expositions rather seem to be of a critical-realistic nature than follow a constructivistic standpoint, as the theory of correspondence is being denied by constructivists.
- The understanding of a modeling domain is ambiguous. At first, the problem of domain notion is examined from a constructivistic viewpoint. "The notion of 'domain' is problematic from a constructivistic viewpoint, since it seems to imply the existence of an objectively, true solution." [Krog95, p. 97]. Later on, this position will be qualified though: "If one use an objectivistic ontology, or one accept a high degree of intersubjective agreement on the modeling domain, this is similar to the

definition of the original framework.” [Krog95, p. 99] The authors seem to mix up the ontological and the epistemological viewpoint. With reality existing independently from the perceiving subject, the assumption that a domain exists independently of a subject is not coherent.

- The evaluation of the semantic quality highlights the problematic interaction between the basic position and the model evaluation on object level. The semantic quality can be understood as the correspondence between domain and model, although, from a constructivistic point of view, this evaluation cannot be applied.

The *Guidelines of Modeling* do not show any inconsistencies. The basic positions are not contrary to the statements for model evaluation.

4.2 Object level: Comparison of the evaluation approaches

The approaches by KROGSTIE ET AL. and MOODY/SHANKS are based on the assumption that quality factors do represent a number of objectives. The semantic quality, according to KROGSTIE ET AL. is composed of the objectives validity and completeness. As can be derived from the meta models, the approach by KROGSTIE ET AL. is assigning exactly one quality factor to each objective. The characteristics of the quality factors as found in the MOODY/SHANKS approach correspond to the objectives which can be found in the approach by KROGSTIE ET AL., i.e. no combination of individual objectives but individual, atomic objectives. Those quality factors (MOODY/SHANKS) or objectives (KROGSTIE ET AL.) should be achieved through measures (activities at MOODY/SHANKS and Means at KROGSTIE ET AL.). Whereas in the MOODY/SHANKS approach one activity can meet several goals, at KROGSTIE ET AL. one individual means is always utilized for reaching one objective. Only in the approach made by MOODY/SHANKS, the interaction between the goals is taken into consideration. According to the GoM, the measures do not depend on a concrete objective.

Concrete standards for evaluating the result types can be found at [KrLS95b], [Mood98] and in the GoM [Schü98]. For delivering a comprehensive evaluation of the objectives, not only would the result types have to be stated more precisely but also the preference types mentioned above: preference of degree, type, safety and time. Only in the approach made by MOODY/SHANKS the problem of weighting within the preference function is made an issue of. The theoretical concept, however, of such a weighting is not described as the “arbitrary evaluation” of the individual user is playing a significant role.

In summary, the following three main aspects can be derived from this comparison:

- A differentiation of objectives and measures is explicitly pointed out in the GoM, at MOODY/SHANKS and at KROGSTIE ET AL.
- According to KROGSTIE ET AL. the type of interrelation between measures and objectives is a critical one, as the authors’ approach is based on the assumption that measures can be definitely allocated to objectives. This allocation of measures to objectives, however, is contrary to the fact that the interrelations between objectives and measures is being determined through the preference functions of the decision maker. In a specific context, for example, a measure can help to reach goal

A whereas, in a different context - due to a different preference function - goal A is of no significance.

- According to MOODY/SHANKS there exist interdependencies between the quality factors. Within the meta model, this is represented by the interrelation of the entity type quality factor to itself. This assumption does not take into account the character of the interdependencies between the objectives. Interdependencies between objectives are no characterizing features of the objectives themselves. It is rather that those interdependencies develop no earlier than in the specific context. The interrelations between the objectives is determined by the form of the preference function (complementary, contrasting or neutral).

In figure 5, the interrelations between measures, preference functions and objectives can be found. As can be derived from this figure, a statement on the measures can only be made in relation to the preference function. The same goes for the objectives as they cannot be examined without looking at the preference function.

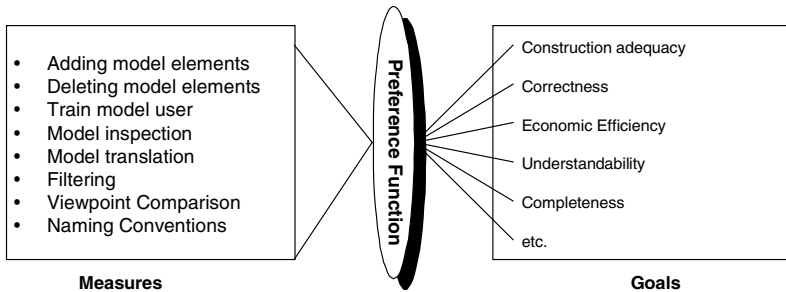


Fig. 5. Decision-oriented interrelation between goals and measures

5 Prospects

When evaluating information models, literature is mainly focusing on discussing criteria suited for evaluating the quality of a model as well as individual modeling measures.

Generally, different kind of quality measures are applied in the phases of the software development cycle, especially in requirements engineering and software design. The questions are, to which implications the usage of distinguished quality measures leads and whether the standardization of quality measures can be useful.

Further on, research work should concentrate on improving the theoretical basis of quality frameworks due to theory-lacks in the field of information model quality:

- Decision-oriented aspects which serve as the basis for evaluation and selection of measures, however, are not being analyzed. Many authors ignore the interdependencies between goals, measures and the preference function of a subject. Especially the unknown preference functions of information modeler and user create the necessity for examining the preference types from an empirically point of view.

Research in this problem area would make good use of a descriptive information modeling research as it is likely that the traditional instruments of empirical research are suited for gaining new insight into this problem area.

- In the arising area of ontologies several publications are discussing the quality of ontologies (see for example the Bunge-Wand-Weber-Ontology as an “Meta-Ontology” [Webe97] and for an ontology in the sense of Artificial Intelligence [FoGr97]). Ontologies can be interpreted as a special case of an information model. So one has to consider, whether the evaluation problems of ontologies differ from the evaluation process of traditional information models.
- From a meta science point of view in regard of ontological engineering the question arises, how the incommensurability and QUINES ontological relativity affect the construction of ontologies.

References

- [Bitz81] Bitz, M.: Decision Theory. Munich 1981. (in german).
- [BuMo79] Burrell, G.; Morgan, G.: Sociological Paradigms and Organisational Analysis. London et al. 1979.
- [DaSh96] Darke, P.; Shanks, G.: Stakeholder Viewpoints in Requirements Definition. A Framework for Understanding Viewpoint Development Approaches. REQUIREMENTS ENGINEERING 2/1996, pp. 88-105.
- [Falk97] Falkenberg, E.D. et al: FRISCO. A Framework of Information Systems. Summary of the FRISCO Report. December 1996 (<http://leidenuniv.nl/publ/rul/fri-w60.zp>, 4.4.1997).
- [FoGr97] Fox, M.S.; Grüninger, M.: Ontologies for Enterprise Modelling. In: Enterprise Engineering and Integration. Building International Consensus. Proceedings of ICEIMT '97, International Conference on Enterprise Integration and Modeling Technology. Hrsg.: K. Kosanke, J.G. Nell. Berlin et al. 1997, S. 190-200.
- [HilK98] Hirschheim, R.; Iivari, J.; Klein, H.K.: A Comparison of five alternative approaches to information systems development. AUSTRALIAN JOURNAL OF INFORMATION SYSTEMS 4/1998. (also <http://www.cba.uh.edu/~parks/fis/sad5.htm>, 16.3.1999).
- [Inwo95] Inwood, M.J.: Weltanschauung. In: Honderich, T. (Ed.). The Oxford Companion to Philosophy. Oxford 1995, p. 909.
- [Krog95] Krogstie, J.: Conceptual Modeling for Computerized Information Systems Support in Organizations. PhD Thesis, University of Trondheim. Trondheim 1995.
- [KrLS95a] Krogstie, J.; Lindland, O.I.; Sindre, G.: Towards a Deeper Understanding of Quality in Requirements Engineering. In: Proceedings of the 7th Conference on Advanced Information Systems Engineering (CAiSE '95). Ed. by J. Iivari, K. Lyytinen, M. Rossi. Berlin 1995, pp. 82-95.
- [KrLS95b] Krogstie, J.; Lindland, O.I.; Sindre, G.: Defining Quality Aspects for Conceptual Models. In: Proceedings of the International Conference on Information System Concepts (ISCO3). Towards a Consolidation of Views. Marburg 1995. Preprint.
- [Laux98] Laux, H.: Decision theory. 4th Ed., Berlin et al. 1998.
- [LiSS94] Lindland, O. I.; Sindre, G.; Solvberg, A.: Understanding Quality in Conceptual Modeling. IEEE SOFTWARE 2/1994, pp. 42-49.
- [Ming95] Mingers, J.C.: Information and Meaning: foundations for an intersubjective account. INFORMATION SYSTEMS JOURNAL 1995, pp. 285-306.
- [Mood98] Moody, D.L.: Metrics for Evaluating the Quality of Entity Relationship Models. In: Conceptual Modeling - ER '98. 17th International Conference on Conceptual Modeling. T.W. Ling, S. Ram, M.L Lee. Singapore, November 1998, pp. 211-225.

- [MoSh94] Moody, D.L.; Shanks, S.: What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In: Loucopoulos, P. (Ed.). Entity-Relationship- Approach - ER '94. Business Modelling and Re-Engineering. 13th International Conference on the Entity-Relationship Approach. Proceedings. Berlin et al.1994, pp. 94-111.
- [MoSh98] Moody, D.L.; Shanks, G.: What Makes a Good Data Model? A Framework for Evaluating and Improving the Quality of Entity Relationship Models. Australian Computer Journal 3/1998, pp. 97-110.
- [MoSD98] Moody, D.; Shanks, G.; Darke, P.: Improving the Quality of Entity Relationship Models - Experience in Research and Practice. In: Conceptual Modeling - ER '98. 17th International Conference on Conceptual Modeling. T.W. Ling, S. Ram, M.L. Lee. (Eds.). Singapore, November 1998, 255-276.
- [Resc97] Rescher, N.: Objectivity. The Obligations of Impersonal Reason. Notre Dame, London 1997.
- [Schü98] Schütte, R.: Guidelines of Reference Modeling. Construction configurative and adaptable models. Wiesbaden 1998. (in german).
- [ScRo98] Schütte, R.; Rotthowe, T.: The Guidelines of Modelling as an approach to enhance the quality of information models. In: Conceptual Modeling - ER '98. 17th International ER-Conference, Singapore, November 16-19, 1998. T. W. Ling, S. Ram, M. L. Lee (Eds.), Berlin et al. 1998, pp. 240-254.
- [ShDa98] Shanks, G.; Darke, P.: Understanding Data Quality in Data Warehousing: A Semiotic Approach. Proceedings of the Information Quality Conference, Massachusetts Institute of Technology. November 1998 (Preprint).
- [ShDa97] Shanks, G.; Darke, P.: Quality in Conceptual Modelling: Linking Theory and Practice. In: Proceedings of the Asia-Pacific Conference on Information Systems (PACIS). Brisbane 1997, pp. 805-814.
- [Steg70] Stegmüller, W.: Experience, Fixing, Hypothesis and simplicity of concept and theory formation in the theory of science. Problems and Results of the theory of science and analytical philosophy, Vol. II: Theory and Experience, Study version Part A, Berlin et al. 1970. (in german).
- [Webe97] Ontological Foundations of Information Systems. Melbourne 1997.

XML-based Components for Federating Multiple Heterogeneous Data Sources

Georges Gardarin, Fei Sha, Tram Dang Ngoc

PRISM Laboratory, University of Versailles-St-Quentin
45 avenue des Etats-Unis, 78035 Versailles Cedex, FRANCE
E-mail: firstname.lastname@prism.uvsq.fr

Abstract. Several federated database systems have been built in the past using the relational or the object model as federating model. This paper gives an overview of the XMLMedia system, a federated database system mediator using XML as federating model, built in the Esprit Project MIRO-Web. The system is composed of four main components: a *wrapper generator* using rule-based scripting to produce XML data from various source formats, a *mediator* querying and integrating relational and XML sources, an *XML DBMS extender* supporting XML on top of relational DBMSs, and client tools including a *Java API* and an *XML query browser*. The results demonstrate the ability of XML with an associated query language (we use XML-QL) to federate various data sources on the Internet or on Intranets.

1 Introduction

In the past few years, research in semistructured data has become a hot topic [1], [2]. Different approaches have been proposed for efficiently managing semistructured data from diverse sources. Significant experiences are the Lore system [13] developed at Stanford University built from scratch for managing semistructured data, Ozone built on top of the O2 object-oriented DBMS, and STRUDEL [8] developed at AT&T for managing web-based data.

In this paper we present our experience in federating multiple data sources using XML and XML-QL, a query language jointly developed by AT&T and INRIA [4]. This work is part of the European ESPRIT project MIRO-Web [7], a federated database system integrating relational, object-oriented and semistructured data. MIRO-Web is federating heterogeneous data sources using a semistructured data model. The model is a slight variation of OEM [13]. Data are exchanged among sources using XML and XML-QL, i.e., a query language based on XML patterns [4]. As repository to store and retrieve XML documents, we use Oracle 8.0, the well known object-relational DBMS.

In the following, we give an overview of our federated database system architecture and describe the main components of the system. The system is now called XMLMedia (for Mediating in XML) and is on his way to be commercialized by a French-based company. Components include an *XML wrapper* using a rule-based data extraction technology developed at GMD IPSI in Germany, an *XML extender* (also called a cartridge) to store and query XML documents in a relational DBMS, an *XML mediator* which receives XML-QL queries from client sites, then decomposes them in either SQL or XML subqueries, routes the resulting subqueries to the source wrapper and assemble the results in XML. Finally, it also includes an *XML query browser* to formulate queries in XML-QL and present the results, plus a *Java DOM API* for submitting queries in Java and manipulating results.

The paper is organized as follows. Section 2 introduces the objectives of the XMLMedia system and gives some background on related projects. Section 3 describes the system architecture and introduces the components. Section 4 details the XML extender for relational DBMSs. Section 5 presents the principles of the mediator. We conclude in Section 6 and summarize the advantages of XML for mediation

2 Objectives and Backgrounds

In this section, we summarize the objectives of our system and present some backgrounds on semistructured databases.

2.1 System Objectives

The goal of the XMLMedia system is to provide in an efficient way integrated access to multiple data sources on the Internet or on intranets using XML protocols and tools based on Java components. The components can be assembled together or used separately in different application contexts. Possible applications are the constitution of electronic libraries from existing sources, the collect of information for trading systems, the assembling of fragments of data for technology watch, and more generally knowledge management and information diffusion in the enterprise. Used together, the components form a unique solution to support iterative definition and manipulation of semistructured databases derived from existing data sources or loosely structured files in Internet/Intranet environments. They are based on the new emerging standard XML and its derivative such as XML-QL, a query language for XML proposed at W3C by AT&T and INRIA.

Today Intranet/Internet platforms support access to existing files, databases, data warehouses, and legacy applications from a single browser interface. However, the existing tools do not allow the integration of multiple data sources, due to the lack of metadata generators, metadata integrators, and distributed query evaluators including update transactions. In such a context, some data sources are well structured, while others are not. The XMLMedia components provide a nice way to merge structured and non-structured data and make it accessible through established standards, namely SQL3, JDBC, and XML, on the Internet or Intranets.

2.2 Related Work

There has been recently a lot of work on semistructured data. Some systems pioneered a storage model based on object DBMSs. TSIMMIS [15], Lore [13], and Strudel [6] store their data as graphs. [16] presents an original approach based on binary relations (one for each label) storing object identifiers and values in the context of the Monet DBMS. They use the MOA algebra to deal with type coercion and object assembling. [3] proposes an object-oriented schema derived from the DTD to store SGML documents. We further detail relational storage techniques for semistructured object in the next subsection.

Experiences in federating data sources through a semistructured model have been conducted in several projects, including TSIMMIS, Lore and YATL. This last system focuses on rule-based translation techniques and object typing [1]. For optimizing queries, several approaches have been studied. Optimization techniques are for example surveyed in [2]. More specifically, [9] optimizes regular path expressions using graph schemas.

The management of metadata is also an important problem in (federated) semistructured DBMS. Originally proposed in [10], strong dataguides can help in query formulation or in query processing [13]. Panoramic dataguide [11] were introduced for query formulation. They are quite similar to path index introduced in object systems. Recently, [12] pioneered the idea of using approximate dataguides, which group together dataguide paths when their target sets are “almost” equal.

2.3 Storing Semistructured Objects in Tables

There are obviously many ways to map graphs into tables. Hence, several mapping schemes are possible for storing semistructured objects in an object-relational DBMS. Our first experiences demonstrate that simple mapping schemes can differ a lot in performance for a given query workload according to very specific detailed choices, such as OIDs assignation, clustering and indexing schemes, number of tables to open, etc. In a first phase, we investigated the following mapping schemes:

- *The bulk data type.* A bulk data type can be used to store directly XML documents with tags and data. Access to individual structural elements is then realized by parsing. This approach is rather efficient for assembling documents, but requires specific index structures for retrieval.
- *The single-edge table.* A table can be used to represent each individual edge of the object network as a tuple, consisting of parent-OID, label, type, order, and child-OID. Conventional relational operators, including select, join, recursive join, project, and aggregate, can be used for evaluating queries over such structures. With appropriate indices for values and links the inefficient scan can be avoided to a large extent. In contrast to using a bulk-data type, the approach is rather efficient for accessing small substructures, whereas the overhead involved in assembling complex documents could be rather high.

- *The multiple-edge table.* This scheme is a variation of the previous one where a generic table is used, but one tuple contains all edges going out from a given node. This requires nested tables. A typical scheme for the generic table consists of pairs parent-OID, array of edges, each edge being described by a label, a type, and a child-OID. A first advantage of this solution is that the order is encoded within the array; also, edges outgoing a given node are grouped together; however, searching in array-valued attributes remain a difficult operation in most object-relational DBMSs.
- *The single-edge attribute tables.* The idea is to store all the edges with the same label in a separate table having the name of the label [16]. This approach can be understood as a variation of the single-edge table but with several edge-tables, one for each edge name. A variant can be obtained by clustering or indexing the single-edge table on the label attribute. The document assembling is difficult with the single-edge attribute tables approach. However, it is efficient for selecting elements of a given label. As manipulating table names is less efficient than manipulating indexed attribute values in most object-relational DBMSs, we prefer the single-edge approach with clustering or indexing.
- *The structured tables with overflow.* An explicit relational schema can be derived from a collection of semistructured objects. The schema can be composed of several tables, each designed to group repetitive structures. STORED [5] proposes a declarative query language to specify the mapping between the semistructured data model and the relational model. The mapping exploits regular patterns that can be found in semistructured data to compose multiattribute tables. Furthermore, an overflow graph is managed to store data that do not fit the relational schema.

2.4 Querying XML Documents

Several query languages have been proposed to query semistructured data and XML documents, among them LOREL, XML-QL and XQL.

The LOREL language [13] was one of the most complete query language built for querying semistructured data, more specifically OEM graphs. LOREL is basically an extension of OQL with extended path expressions. A simple path expression is a root name followed by a sequence of tags, for example `Guide.Restaurant.Adress.Street`. This expression refers to all objects reachable from the root following the edges with the given tags. Path expressions can be used to specify collections and in place of attributes for conditions and projections. They are generalized to express search patterns with string profiles, disjunction between tags, optional tags and regular expressions with the Kleen star, allowing 0 to N repetition of a tags. For example, the following request retrieve all attributes of restaurants in cities or streets containing the substring “puy” :

SELECT GUIDE.Restaurant.#

WHERE Guide.restaurant.Adresse.(Ville|Rue) **LIKE** "%puy%".

LOREL is powerful and elegant, but requires the support of OQL, which is quite complex in federated database systems.

XML-QL is another query language specifically designed for XML document collections. XML-QL is based on XML patterns, i.e., XML documents with elements replaced by variables and predicates. XML-QL provides facilities to filter document collections, but also to construct query results as new XML documents. A typical query is of the form :

WHERE <template>

CONSTRUCT <template>.

A template can be either an XML document with variables and conditions including joins, but can also be a condition with regular expressions or function calls, including Skolem functions to create new nodes. A template can also be an XML-QL query, which allows to nest queries and express powerful constructs. A simple example of an XML-QL query is :

WHERE

```
<Guide>
  <restaurant> r
    <Address>
      <City> c </City>
      <Street> s </Street>
      (c LIKE "%puy%" OR s LIKE "%puy%")
    </Address>
  </restaurant>
</Guide>
```

CONSTRUCT

```
<Guide> r </Guide>
```

A derivative of such languages called XQL is under a standardization by the W3C. The language is similar to XML-QL but has a different rather complex syntax. The basic idea is to extend URLs and hierarchical designators to represent queries. Special characters are introduced to encode search patterns, such as / to look for the child of a node, // to look for all descendants, * to designate any label, @ to capture an attribute, [...] to nest sub-queries, ? to return a node and ?? to return a node and all its descendants. XQL could be part of the XSL standard or a stand alone component.

3 System Architecture and Components

In this section, we present an overview of the system architecture and summarize the functions of the main components.

3.1 Architecture Overview

As mentioned in introduction, the XMLMedia system federates multiple data sources using a semistructured data model and XML as the basic exchange vehicle. The system follows the DARPA I3 architecture including three layers of functions : the translation layer, the mediation layer and the coordination layer. The overall system architecture is represented in figure 1. It is composed of Java packages constituting components (they should be arranged as JavaBeans). For accessing relational DBMSs, we use JDBC. For the rest, we developed our own components.

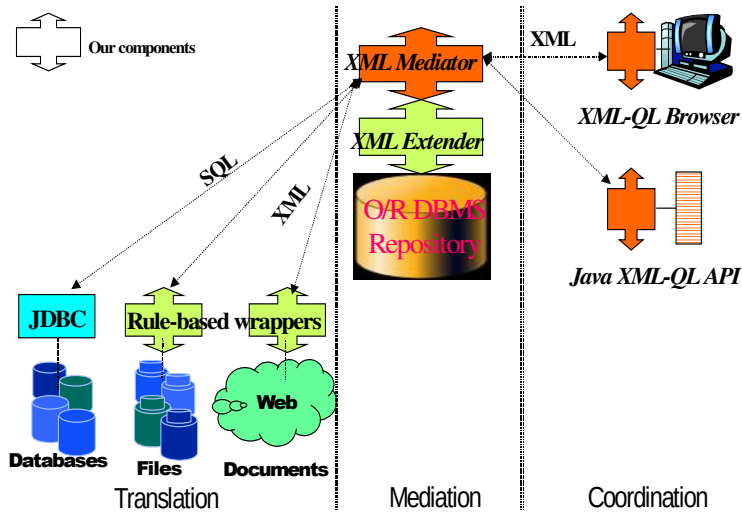


Fig. 1. System Architecture

3.2 The Three Tiers

The system follows a three-tiered architecture, with clients, application servers running the mediating components, and the data tier including most of the translation services. More precisely, we distinguish three layers as follows :

The **translation layer** includes the data sources and wrappers to wrap them in XML or SQL. XML wrappers are built from the JEDI toolkit developed at GMD in Germany [8]. JEDI is a powerful tool based on production rules. The rule allows the wrapper implementor to write search patterns in source documents and to produce variables. The variables can later be bundled in XML. Rules can call each other and can be recursively applied. Different formats of sources are supported including text, HTML, RTF, PS, DOC, XML, etc. Queries can be expressed against the source using a subset of OQL.

The **mediation layer** includes the key components we will describe in the two next sections. The XML Extender extends any JDBC compatible DBMS to support XML

document storage and querying. This component provides the necessary tools to store efficiently XML in relational tables and to run SQL3 queries extended with generalized path expressions. The query language is similar to LOREL, but built on SQL. Also, it allows result restructuring through a special AS clause. The XML Mediator does mediation between all the sources, including XML and SQL ones. It processes XML-QL queries, divides them in sub-queries according to the source capabilities, collects the results in XML, assembles them, and sends an integrated answer to the client.

The **coordination layer** includes all the client tools. It is composed of an XML-QL browser to formulate queries and present results, and of a JAVA API. This API allows Java programs to send XML-QL queries and to get the results as an XML document. The document can further be exploited in Java using the DOM tools.

4 The XML Database Extender

In this section, we describe the XML Database Extender (sometimes also called cartridge), which is able to store XML documents in any object-relational DBMS and which provides an extended SQL3 with path expressions to retrieve XML document fragments and reassemble them.

4.1 XML Document Modeling

As in many research projects, we use labeled directed graph for modeling semistructured object. Our modeling is an ordered version of the OEM model [13] extended with different edge types. A document consists of a collection of objects, in which each object is either complex or atomic. A complex object is an optionally ordered set of <name, object> pair. An atomic object is a value of type int, real, string, image, video, and more generally of any type supported by the underlying object-relational DBMS (user data types are possible). The optional order gives the relative order of edges among those going out from a given node. To better model related XML documents, we distinguish two types of edges, as shown in Fig. 1: an edge originated and targeted within the same document is called an *aggregation link*, and an edge between different documents is called an *association link*. Hence, a document is a graph, with edges labeled by names and leaves labeled by values. Documents can be linked together by association links.

When assembling a document, only links of aggregation type are considered as relevant. This information is particularly important to model XML documents, which includes M to N relationships (i.e., association links). The order is simply memorized through an ordinal number. This is important to assemble XML document items in the correct order. Figure 2 shows on an example how we model XML linked documents.

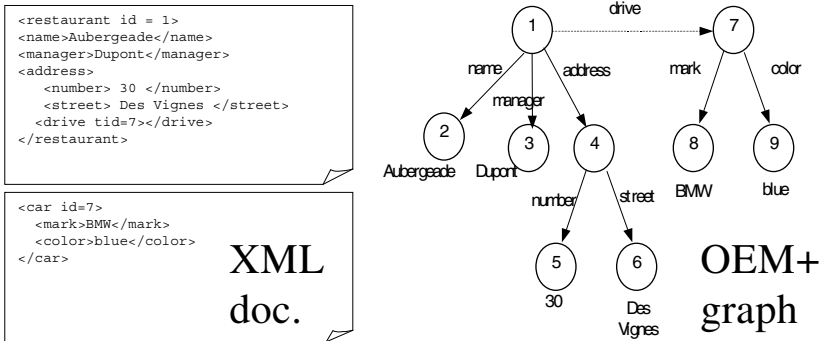


Fig. 2. XML document modeling

4.2 Storing Documents in Tables

For simplicity and efficiency in retrieval, we selected the single-edge table approach. Edges in the graph are stored in a table called the *edge table*. In the database, each node is identified by a unique identifier. For allowing efficient clustering of the document edges, the root node identifier of each document is repeated as the first attribute of each edge of the document. Hence, each edge is represented by a flat tuple [root, origin, target, label, link type, node type, order]. The root attribute indicates the object root OID from which the edge is a descendent. The origin and target attributes memorizes the node OID of the origin and target node respectively. Label is the label of the link. In the edge table both *aggregation* and *association* links are stored. Hence, link type indicates whether a link is a composite aggregation or an association. Node type is the type of the value attached to the target node. For internal nodes, it is simply “NODE”, while it is the name of the leaf table containing the value for a leaf node. The structure of the edge table is exemplified in Figure 3. To group edges of a document in one or more pages, the *edge table* is clustered on root_oid. A clustered index is maintained by the RDBMS to give direct access to edges of a document, e.g., to join efficiently with a relational table.

root	origin	target	label	ltype	ntype
1	1	2	name	AGGR	STRING
1	1	3	manager	AGGR	INTEGER
1	1	4	address	AGGR	NODE
1	4	5	number	AGGR	INTEGER
1	4	6	street	AGGR	STRING
1	1	7	drive	ASSC	NODE
7	7	8	mark	AGGR	STRING
7	7	9	color	AGGR	STRING

Fig. 3. An instance of the edge table

Leaf nodes with their values are stored in *leaf tables*. By default, one leaf table is used to store leaves of each type. Special mapping specifications can be used to define target tables for several leaf nodes. For the time being, the mapping language allow the administrator to stipulate which XML element is going to be stored in a table. The element has to be a tuple conforming to the value table schema. We envision to extend the mapping language for mapping complex values in a unique object-relational table in the future.

As mentioned above, this storage structure is quite efficient for query processing. Indices can be declared on origin and taget OIDs and labels, to accelerate selection and join when querying. The problem with this approach is the cost of assembling full documents data from the leaf tables. As tuples are randomly distributed within the leaf table, assembling a complex document may require several I/Os in the leaf table. To reduce the high cost of assembling document values, we use a clever node OID allocation policy. A unique integer value is assigned to each node as its identifier using a sequence number (provided by the RDBMS). Within an XML document, ascending node OIDs are assigned to each node in the breath-first traversal order. Thus, identifiers of leave nodes of a document are close together. We cluster leave tables on their identifiers and index them on that identifier. To retrieve the data of a document, range queries can be used in each leaf table containing data from that document. Figure 4 illustrates a simple leaf table for storing the string data type values. Notes that leaf tables are clustered on NODE ID, to group chunks of documents together.

No- deID	Value
3	"Dupont"
6	"des Vignes"
8	"BMW"
9	"blue"

Fig. 4. Example of a leaf table for string values

4.3 Document Query Processing

As already mentioned, we use an extended SQL to query XML documents. The extension for SQL is similar to LOREL for OQL, i.e., we mainly support generalized path expressions in SQL queries. Generally, relational systems are considered inefficient for processing queries containing path traversals. However, in the context of semistructured data, some characteristics of relational systems can compensate the handicap. First, in object-relational systems, path traversals can be processed in a set-oriented manner using successive joins on indexed attributes (the source and target OIDs). Second, in systems supporting recursive queries, path expressions can be processed using them. For example, in Oracle 8, the hierarchical recursive query is well suited for evaluating path expressions.

Obviously, the cost of path traversals increases as the path length increases. To optimize long path traversals, our implementation support the concept of *path index*. A

path index is simply a join index of the edge table with itself, using target equals origin as join predicate. The path name is kept by concatenating the edge names. By joining again the path index with the edge table, we can generate path index of length 3, and so on up to path index of length n . Path index tables can become large. To reduce the size of path index tables, we can choose to store only paths frequently occurring in the graph. A no match will then requires normal processing against the full edge table. A path index reduces the number of joins or level of hierarchical traversal in processing path expressions. To traverse a given path expression, we first search within the path index corresponding to the maximum covered path length, and then follow up within the edge table. A path index of length n cannot be used directly to evaluate path expressions of length shorter than n .

In semistructured databases, regular path expressions with Kleene stars have special importance. Semistructured data have irregular and changing structures. By means of regular path expressions, users can query the database without the full knowledge on data structures. Although convenient to use, regular path expressions in general cannot be evaluated directly due to their prohibitive cost. Up to now, the proposed solutions for optimizing regular path expressions are all based on pruning techniques [9]. To apply pruning, some sort of metadata is required. Several variations of the representative objects [14] and dataguides [10] including the approximate dataguides [12] have been proposed. They are useful to keep track of the possible nesting of labels and avoid full scan of the database.

In our system, any label can be accessed directly regardless the distance from the root. This is due to the existence of an index on the labels. In addition, the graphs can be traversed in forward or reverse direction through joins. The traversal can be efficient if path index are maintained, as explained above. These features are quite helpful for processing path expressions with wildcard and more generally Kleene stars.

To further accelerate path expression processing, we maintain a specific sort of metadata called the *longest path set*. Informally, this is the set of longest paths in the graphs. Queries based on path expressions search for nodes but are formulated from labels. Hence, in our structure, we keep only labels. We model the possible paths as a regular language where labels are letters from an alphabet, and where regular path expressions are words (infinite words in the event of loops in the graph). The goal being to determine words that belong to this language, we do not keep words that are factors of a longer word. Regular path expressions have the following format: $l_o l_1 \dots l_i \dots l_j \dots l_{j+1} \dots l_n$, where the Kleene star matches any path of any length. The goal of the rewriting is to find all sub-words from the set of words in the longest path set which start with $l_o l_1 \dots l_i$ and finish with $l_j l_{j+1} \dots l_n$. Then, the query can be transformed in a query with a disjunction of fully documented path expressions in qualification.

In summary, we expand regular path expressions using the longest path set words and sub-words. For simplicity, we assume that graphs have no loop. It is possible to handle loops, but they have to be detected at document loading and expanded with special labels to encode infinite longest paths.

5 The XML Mediator

The XML mediator is the other key component of the mediation layer. Through the browser, the client issues an XML-QL query to the mediator and waits for an answer. As usual in mediation systems, the mediator accepts the query and decomposes it into subqueries, one for each source, with in addition a recomposition subquery. Subqueries are routed to a gateway in charge of shipping the query to the appropriate wrapper. The gateway is able of managing a pool of data source connections to save overhead. When the connection has been allocated, the gateway issues the subqueries to the wrappers and waits for a response. The wrappers process the subqueries by consulting the associated data sources and generate subanswers that are returned to the mediator. The mediator combines the subanswers by using the composition subquery and generates the final answer that is returned to the client

In addition, the mediator provides some facilities to translate queries according to the capabilities of the wrapper. To perform this translation, specialized components called *doors* are included. A door is a generic class and can be extended by query and result translation methods. Figure 5 gives an overview of the mediator architecture.

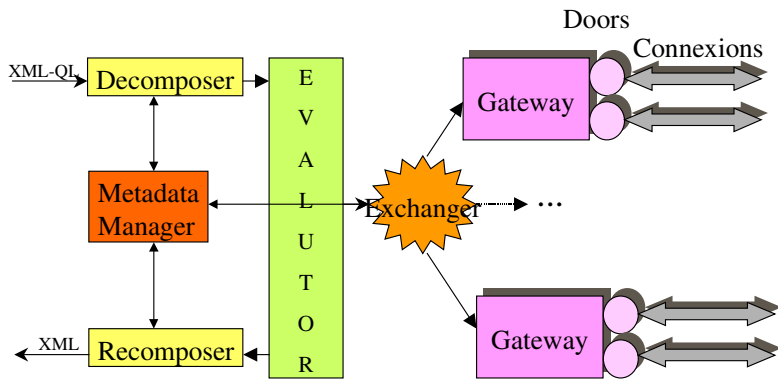


Fig. 5. The mediator architecture

The role of the main components of the mediator can be summarized as follows :

- The Decomposer decomposes XML-QL queries in query graphs.
- The Evaluator identifies local subqueries to ship and run the recomposition query.
- The Exchanger routes the query to the appropriate gateway and collects the result back.
- The Gateway manages connection pools to data sources and submits subqueries to wrappers.

- The Door translates queries for local wrappers and translates answer back in OEM graphs
- The Recomposer recomposes an XML answer for the client.
- The Metadata manager collects and maintains schemas and label paths for each data source.

In summary, the XML mediator is somehow similar to a relational mediator except that it is based on XML-QL and returns data in XML. The semi-structured OEM+ model supported by data and query graphs is the basis of our internal implementation. Thus, XML data are internally represented as extended OEM graphs similar to that manipulated in the XML extender described in the previous section. Queries are also represented as graphs with constraints (i.e., restrictions) and links (i.e., joins). Extended OEM graphs support easily the mixing of relational data and true semi-structured data provided by XML wrappers or SGML databases.

6 Conclusion

In this paper, we have presented XMLMedia, a mediating system based on XML. XMLMedia federates both XML documents and SQL tables and unify them in a virtual XML digital library. The XML Extender provides the necessary fonctionnalties to store and manipulate XML documents on top of an object-relational DBMS. Coupling the capabilities of the extender with multimedia features now available in most object-relational DBMSs gives powerful functions to retrieve documents by contents, e.g., using similarity search on keywords or image features. The mediator offers the XML-QL query language to query the virtual libraries of XML documents.

A first version of the system is already experimented in several applications as a test-bed, including a tourism virtual Web site in Tyrol, an Intranet in a Spanish hospital and a trading application. Performance is achieved by the nice features well established of object-relational systems and by appropriate pools of buffers and connections. The mediator can also be used to feed Web data in a datawarehouse and more generally to collect and disseminate information in the enterprise. A demonstration of the system can be found on the Web at <http://nora.prism.uvsq.fr/miroweb/>.

Our experience in federating data sources through XML and XML-QL demonstrates the advantages of the semistructured data model for exchanging data. First, it is flexible and auto-documented. No type requires to be predefined, as with the object model for example. Second, it is easy to merge semistructured data and relational one through direct mapping, which can be more or less sophisticated for optimization purpose. Third, XML is an appropriate exchange model on the Web and writing wrappers is an easy task, especially with a powerful wrapper generator as is JEDI. Fourth, XML-QL is appropriate for client-server dialog, as it is XML uniform (a query can be seen as an XML document). Fifth and it is probably the most important, XML gives

good opportunities to unify classical database processing with information retrieval approaches. The power of the resulting tool is not well evaluated yet.

Acknowledgments

We would like to thank Peter Fankhauser, Dana Florescu, Henri Laude and Patrick Valduriez for helpful discussions, particularly in the context of the MIRO-Web project. We owe them many ideas.

References

1. Serge Abiteboul, “*Querying Semistructured Data*”, in Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, 1997
2. Peter Buneman, “*Semistructured Data*”, in Proceedings of the 16th Symposium on Principles of Database Systems, Tucson, Arizona, 1997
3. Vasilios Christophides, Serge Abiteboul, Sophie Cluet, Michel Scholl, “*From Structured Documents to Novel Query Facilities*”, in Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, USA, May 1994
4. Alin Deutsch, Mary Fernandez, Dana Florescu, Alon Levy, Dan Suciu, “*XML-QL: A Query Language for XML*”, 1998, <http://www.w3.org/TR/NOTE-xml-ql/>
5. Alin Deutsch, Mary Fernandez, Dan Suciu, “*Storing Semistructured Data with Stored*”, ACM SIGMOD International Conference on Management of Data, SIGMOD Record Vol. 28, N° 2, Philadelphia, June 1999.
6. Peter Fankhauser, Gerald Huck, “Cooking XML : An Extraction Tutorial for Jedi, GMD, Darmstadt, April 199, <http://www.darmstadt.gmd.de/~huck/jedi/tutorial>
7. Peter Fankhauser, Georges Gardarin, Moricio Lopez, J. Muntz and Antony Tomic, “*Experiences in Federated Databases: From IRO-DB to MIRO-Web*”, in Proceedings of the 24th International Conference on Very Large Data Bases, New York, USA, August 1998
8. Mary Fernandez, Daniela Florescu, Alon Levy and Dan Suciu, “*A Query Language for a Web Site Management System*”, SIGMOD Record, vol. 26, no. 3, pp. 4-11, 1997
9. Mary Fernandez and Dan Suciu, “*Optimizing Regular Path Expressions Using Graph Schemas*”, in Proceedings of the 14th International Conference on Data Engineering, Orlando, Florida, USA, February 1998
10. Roy Goldman and Jennifer Widom, “*DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases*”, in Proceedings of the 23rd International Conference on Very Large Data bases, Athens, Greece, 1997
11. Roy Goldman, Jennifer Widom, “*Interactive Query and search in Semistructured Databases*”, International Workshop on the Web and Databases, Valencia, Spain, March, 1998.
12. Roy Goldman, Jennifer Widom, “*Approximate Dataguides*”, International Workshop on Query Processing for Semistructured Data and Non-Standard Data, Jerusalem, Israel, January, 1999.
13. Jason McHugh, Serge Abiteboul, Roy Goldman, Dallen Quass, Jennifer Widom, “*Lore: A Database Management System for Semistructured Data*”, SIGMOD Record, 26(3): 54-66, September 1997

14. Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe, "*Representative Objects: Concise Representations of Semistructured, Hierarchical Data*", in Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, 1997
15. Yannis Papakonstantinou, Hector Garcia-Molina, Jennifer Widom, "*Object Exchange Across Heterogeneous Information Sources*", in Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, 1995
16. Roelof van Zwol, Peter Apers, Annita Wilschut, "*Implementing Semi Structured Data with MOA*", International Workshop on Query Processing for Semistructured Data and Non-Standard Data, Jerusalem, Israel, January, 1999.

A Process-Integrated Conceptual Design Environment for Chemical Engineering

Matthias Jarke, Thomas List, Klaus Weidenhaupt

Lehrstuhl für Informatik V, RWTH Aachen
Ahornstr. 55, 52072 Aachen, Germany
jarke@informatik.rwth-aachen.de

Abstract. The process industries (chemicals, food, oil, ...) are characterized by - continuous or batch -- processes of material transformation. The design of such processes, and their mapping to the available equipment (plants composed of production units in which reactions take place), is a complex process that determines the competitiveness of these industries, as well as their environmental impact. In cooperation with researchers and industry from chemical engineering, we have developed the idea to capture and evaluate the experiences gained about process designs in so-called *process data warehouses*. The data sources for such process data warehouses are highly heterogeneous tools, e.g. for conceptual design (termed flowsheeting in chemical engineering), for mathematical simulations of large non-linear differential equation systems, for measurements gained with experimental usage of equipment at small scale or in original size, or even from molecular modeling. The clients of a data warehouse are interested in operational data transfer as well as experience analysis (pattern detection, process mining) and reuse.

Starting from an empirical analysis of the requirements for a process data warehouse, the paper describes the solution architecture we are pursuing, the models driving the approach, and the status of a prototypical implementation we are undertaking. The prototype links commercial components operationally through advanced wrapping techniques, where the workflow is determined by constraint analysis in a logic-based meta model and executed through a process-integrated modeling environment. In the conclusions, we point out what can be learned from this work for conceptual modeling in general.

1 Introduction

Chemical engineering is the combination of physical, chemical, biological, and informational operations on a chemical plant with the aim of transforming input materials in a manner, that a material product with desirable properties concerning type, behavior, and composition results. The chemical engineering process is, besides this main stream analysis, heavily influenced by considerations of cost and time, but also by environmental side effects, such as energy consumption, detrimental side products, water heating or pollution, and the like.

Since 1997, the German national science foundation (DFG) is funding a Collaborative Research Center at RWTH Aachen in order to address a coherent solution to the issue of collaborative computer-supported chemical engineering [NaWe99]. This center, called IMPROVE after the German title acronym, focuses on the early phases of the development and re-engineering of chemical processes, the so-called conceptual design and the basic engineering. Decisions made at this stage are known to fix already 80% of the overall costs for investment and overall operation which typically run to hundreds or thousands of millions of EUROS per plant. The improved interoperability of mathematical simulation tools for this sector alone has been estimated to result in savings of about 5bn EUROS for the European chemical industries. The European process industries have therefore, in parallel to the start of IMPROVE, embarked on a standardization effort for process simulation in the BRITE-EURAM project CAPE-OPEN [Bra*99].

As stated, IMPROVE aims at linking the existing island solutions for specific chemical engineering tasks into a coherent engineering process. It combines, on the one hand, different specialist areas such as chemical industries and their users e.g. in the plastics construction sector, and – on the other hand – different tasks such as methods for individual workplaces with those for management tasks. Figure 1 summarizes the overall research design of IMPROVE, a brief overview follows.

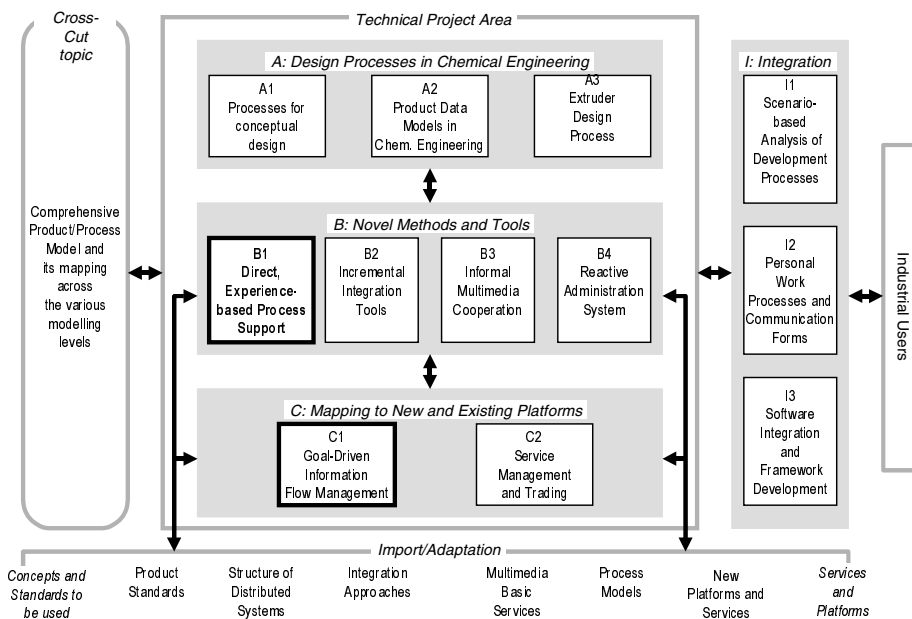


Fig. 1. Structure of the Collaborative Research Center IMPROVE

A pre-requisite is detailed knowledge of individual development processes and their interplay, including interdependencies between product and process. Within IMPROVE, this is the task of the chemical engineering groups (project area A).

This research results in an “idealized” product and process model for cooperative engineering which, however, is neither globally valid nor static but must be customizable to the variety and dynamics of the engineering processes. It must also take into account existing (legacy and new) design and simulation tools. Within project area C, this problem is addressed by developing a so-called process data warehouse and some interoperation service management facilities based on CORBA. Here, we also find the linkage to the above-mentioned CAPE-OPEN effort.

Between these two areas, project area B investigates novel computer science concepts and tools for the different tasks in individual work and cooperation for chemical engineering: direct process support by tracing and reusing developers work expertise; indirect process support by ensuring structure and consistency constraints between the different products; informal, multimedia cooperation to assist awareness and decision making; and project coordination by a flexible and reactive administration system.

The tools supporting these new functionalities, typically on top of wrapped legacy software, must be closely interwoven with each other, without falling into the trap of monolithic integration that has already prevented the success of the CIM movement. The linkage between the A, B, and C layers is instead provided, on the one hand, through metadata management concerning the many aspects of the product and process models (shown on the left of figure 1), on the other by research on an overall framework consisting of actual development scenarios for validation at the level of chemical engineering reality (I1), social science and user interface evaluation (I2), and software architecture (I3).

This paper focuses on *direct process support* for the cooperative development of chemical processes based on *recorded experiences* (project B1 in fig. 1), and on its underlying support by a *process data warehouse* and its meta database, managing the product and process model (project C1). In section 2, we describe these aspects in more detail and point out the central role of flowsheets as a kind of master document in chemical engineering environments. In section 3, the data model of the process data warehouse is presented and linked to recent work in data warehouse and traceability research. Section 4 discusses how engineering environments can be realized on top of such a data model, based on advanced wrapping techniques required for full process integration. These wrapping techniques are demonstrated with the most important new tool for direct process support, a process-integrated flowsheet editor we have developed on top of the widely used commercial tool, VISIO. In section 5, we illustrate how conceptual knowledge, accessible via the process data warehouse, can enhance process-integrated engineering work by method consulting. Section 6 concludes the paper by positioning our work in the conceptual modeling field.

2 Process Integration and Process Data Warehousing

In this section, we first give an overview of our approach to process integration and process data warehousing. Based on an empirical study in a large chemical engineering department, the so-called flowsheet has been identified as the focal concept in such development processes; we therefore briefly describe the state-of-the-art and then point out the advantages of extending flowsheeting with our approach.

2.1 Process-Integrated Design Environments

In contrast to coarse-grained project coordination support at the management level, direct process support aims at fine-grained method guidance for developers at the technical workplace, typically across multiple heterogeneous software tools. Figure 2 gives an overview of the resulting interaction between process-integrated tools (on the right) and process data warehouse (on the left) in our approach.

Direct process support is based on explicit and formal models of the design process. They are composed of working and decision steps performed in the various technical software tools. Just like the early phases in software development, the design of a chemical plant is a highly creative task which cannot be fully and coherently prescribed. Nevertheless, this does not necessarily mean that no process support is possible at all. There exist certain well-understood activities [JaMa96; Döm*96; Lohm98] which occur in many development processes and, therefore, can be generally defined as so-called *process fragments*.

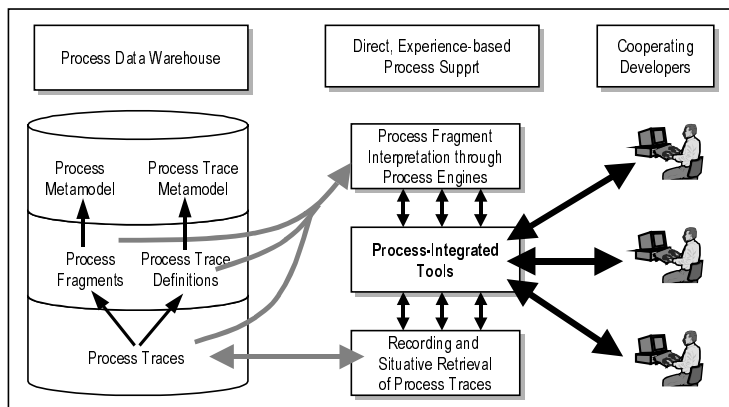


Fig. 2. Basic structure of direct, experience-based process support

Direct process support becomes visible to the user by *process-integrated behavior* of his/her interactive design tools. By process integration [Poh*99] we mean the dynamic adaptation of tools according to the enactment of process fragments. Such process fragments may include: *process automation* by the automated invocation of single tool actions through the process engine, but also *process enforcement and guidance* by restricting user access to certain tool functionality (e.g. by deactivating corresponding menu points) or by highlighting the currently relevant design objects in the user interface of the tool according to the current state of process enactment.

2.2 The Process Data Warehouse

Currently, development processes in chemical engineering are ill-documented, let alone standardized. This complicates the identification and formalization of generally definable, fine-grained process knowledge significantly. Process support has, there-

fore, to start from concrete experiences within and across development processes. A prerequisite is the recording and situated retrieval of experience data as so-called *process traces*.

Process-integrated tools trace their use in the process data warehouse, e.g. by storing the results of tool actions or by recording the user choices among a set of design alternatives. When performing similar processes, the experience knowledge stored in the process traces can be visualized and reused according to the current process situation.

Physically, the three layers of the process data warehouse shown on the left of figure 2 distributed across several subsystems. The upper two layers constitute a *meta database* of knowledge about process definitions and generalized experiences which can be re-used as method advice or consistency control, when activated from the process-integrated tools. We call this the method advisor function of the process data warehouse. In IMPROVE, it is implemented using the ConceptBase system [JGJ*95].

The lower two layers (overlapping in the middle with the meta database) involve massive amounts of trace and product data, often according to task-specific or legacy data models which do not necessarily match the upper-level integrative models perfectly. These data are typically managed in files or, at best, in relational or object-oriented data models.

Direct process support requires a tight integration of the process model interpretation with the design tools used to perform the processes. This poses high demands on the openness of the tools to be process-integrated and causes certain additional effort for the development of suitable wrappers. Our research has, therefore, focused on those parts of the IMPROVE-scenario which can profit most from direct process support.

2.3 The Flowsheet as a Cornerstone of Chemical Engineering Processes

During the development of a chemical process, many information pieces are created which have to be maintained and kept easily accessible. Among these documents, *flowsheets* (graphical representations of function, physics, and structure of a plant), play a prominent role. The importance of flowsheets is not only stressed in various text books about chemical engineering (e.g. [Doug88; Blas97]), but can also be observed in chemical engineering practice.

Indeed, one of the key results of a workshop we conducted in early 1998 with developers and managers from a large chemical engineering department was that the flowsheet reflects in a natural manner the assumptions made by the various stakeholders (chemical engineers, material

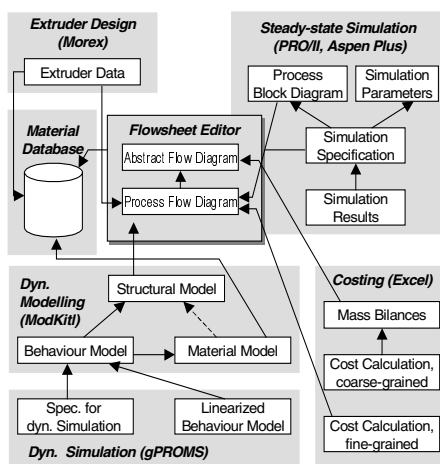


Fig. 3. Role of the flowsheet in the overall IMPROVE scenario

engineers, costing people, safety engineers, managers etc.) about the current state of plant design. The flowsheet hence acts as the main communication medium across several organizational units, and throughout the often decade-long lifecycle of a chemical plant or chemical production process. Moreover, the flowsheet is used as an anchoring point and structuring device for information pieces such as simulation specifications and results, cost calculations, design rationales, safety considerations etc. Managers assess the work progress by the development state of the flowsheet.

The manifold relationships to other information units are illustrated in fig. 3, which depicts the documents and tools used in the IMPROVE demo scenario. They closely correspond to the (unfortunately confidential) ones found in the industrial case study.

2.3.1 *Flowsheet tools in chemical engineering practice*

In current practice, flowsheets are frequently created using drawing tools or CAD systems. These tools provide no specific support for chemical engineering, and often confront the user with superfluous functionality.

In competition to these pure drawing tools, dedicated tools for chemical engineering, such as block-oriented simulators (Aspen Plus, PRO/II), have been augmented by flowsheet user interfaces. This pragmatic trend reflects the close relationship between flowsheet design and mathematical models, and provides the user with considerable support – as long as he/she does not have to leave the boundaries of the tool. The enrichment of simulation programs with flowsheet functionality has indeed led to monolithic, hardly maintainable software systems, which rarely provide open interfaces for extensions. As a consequence of such islands of automation, flowsheet information is repeatedly entered manually, e.g. if different simulators are used within one project. Moreover, as flowsheet editors of common simulators do not allow to annotate individual flowsheet elements with, e.g., cost calculations or safety remarks, isolated flowsheet documents emerge in these working environments, too.

2.3.2 *The CAPE-OPEN Initiative*

These problems have gotten to the point where the monolithic solutions have also led to commercial monopolies: a few vendors control access by chemical companies to progress in the field of chemical engineering research. The CAPE-OPEN project has been a first answer of the chemical industries to these issues. In CAPE-OPEN, the chemical companies BASF, Bayer, DuPont, and ICI, and the oil companies BP and Elf joined to define interoperability standards for chemical engineering software (esp. Simulators) in order to allow interoperation between the tools of the three market-leading vendors as well as to enable the addition of novel methods and tools from small vendors, research groups, and in-house developments.

Due to uncertainties in the underlying software standards, the CAPE-OPEN standard had to be developed in two parallel versions, one in DCOM, the other in CORBA. To manage coherence between the two versions, we had to resort to UML as a higher-level specification, which was developed following a use-case approach [Jar*99]. The interoperation use cases are a nice example of the richness found in the chemical engineering application; we therefore reproduce the hierarchy by which we structured the roughly 160 use cases, in figure 4.

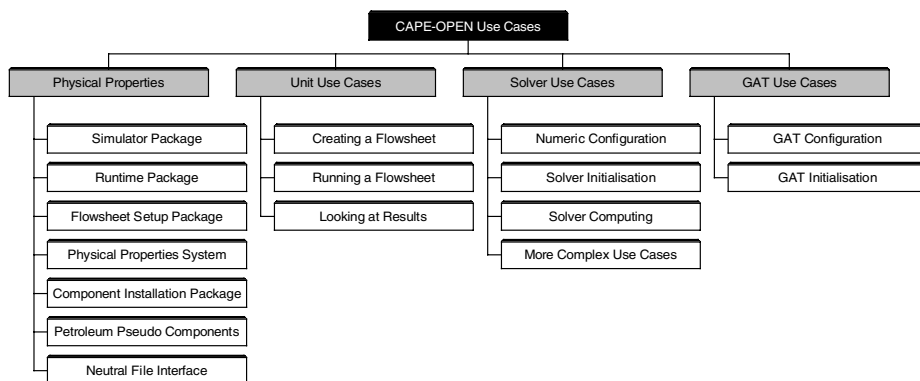


Fig. 4: CAPE-OPEN use case hierarchy for chemical engineering simulation

2.3.3 Flexibility through process integration

The fact, that the flowsheet is involved in manifold development activities, leads to conflicting requirements: from the user's perspective, the flowsheet should be seamlessly integrated into the various process chains (simulation, costing, safety engineering ...) and the corresponding tools. But from a software architecture perspective, tight coupling between tool functionality of different working domains should be avoided in order to remain sufficiently flexible.

Process integration offers the potential to couple different tools more flexibly and to provide high quality support for the user at the same time. Explicit, easily modifiable process fragments guide the user during activities across multiple tools, while the tools themselves only cover a limited scope. In this way, process integration complements data integration mechanisms, which maintain structural consistency between documents of different tools, and component-based approaches such as CAPE-OPEN.

3 The Process Data Warehouse Data Model

The design of the process data warehouse builds on ideas found in data warehousing and software engineering. A major influence has been the separation of logical and conceptual perspectives in source integration for data warehouses, proposed in [CDL*98] and adapted for metadata management in [JJQV99]. For our present purposes, the key argument is that, in highly heterogeneous environments such as chemical engineering, it makes little sense to go for a direct integration of the information sources at the logical level. Instead, as proposed in the Information Manifold project [LeSK95], the information content of these sources should be captured at the level of conceptual modeling and related to an "idealized" *enterprise conceptual model* in order to establish the coverage of the sources, as well as their inter-relationships. IMPROVE follows this approach by adapting a conceptual modeling language proposed for chemical engineering, for the purposes of meta data management and knowledge representation in the process data warehouse.

3.1 The IMPROVE Conceptual “Enterprise” Model

The definition of IMPROVE’s conceptual product data model pursues a strategy which is complementary to the idea of process integration. Based on the chemical engineering data model VeDa [MaGG93], the product data model is divided into logical clusters, which represent one distinct activity within the design process [BaSM99]. They can be interpreted as the perspectives of different designers from several technical disciplines on the product data. A meta model expresses commonalities between the individual partial models. Relationships between objects of different partial models are expressed by dependency links. Our current ConceptBase implementation of this “conceptual enterprise model” comprises roughly 150 metaclass definitions. In terms of [CDL*98], this could be considered the conceptual source models. The “logical source models”, then, correspond to the data models of individual chemical engineering tools. Note, however, that when re-using legacy tools, their data models will usually cover aspects of more than one conceptual source model.

The partial models *Chemical Process* and *Plant*, which are closely related to the concept of flowsheet, represent two key aspects of the product data model. In the following, we discuss two important requirements which arise from these partial models: the treatment of complex refinement structures and a rich type system of flowsheet components.

3.2 Complex Refinement Structures

Flowsheets are refined across various abstraction levels and depicted in specific representation formats. In the context of the IMPROVE project, we are mainly interested in two of the three commonly used flowsheet variants¹: the abstract flow diagram (AFD) and the process flow diagram (PFD).

The AFD specifies the overall function of the chemical process and decomposes it into sub-functions and connections between the sub-functions. The AFD does not yet determine the physical realization of a functional element, but only refines to the level of so-called unit operations, e.g. material conversion, separation, or merging. Functional elements are graphically depicted as boxes, material streams as directed arrows. The PFD is used to represent the plant at the equipment level where certain apparatuses and machines realize the unit operations of the functional structure defined in the AFD. In the PFD, abstracted symbols of the equipments are used in graphical representation.

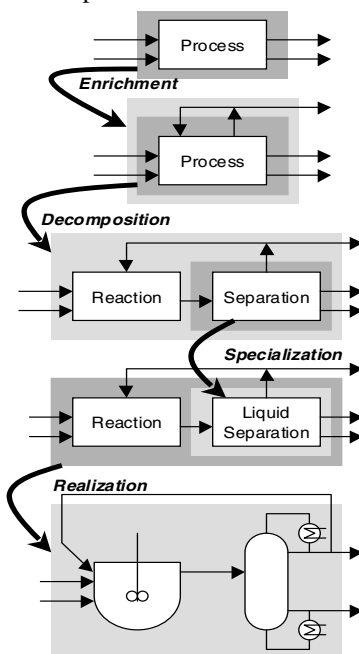


Fig. 5. Types of Flowsheet Refinement

¹ The pipe-and-instrumentation flow diagram is mainly used for the detailed dimensioning of plant equipment which is beyond the early phases of process design, i.e. the scope of the IMPROVE project.

Fig. 5 illustrates the stepwise refinement of the process structure in the AFD and its rough physical realization in the PFD. Even this small example gives an impression of the emerging complex refinement structures, which are characterized by manifold refinement relationships (enrichment, decomposition, specialization, realization) between flowsheet parts.

A „correct“ refinement has to consider a set of consistency constraints, e.g. balancing rules for the in- and out-streams or type consistency between refined and the refining flowsheet parts. A flowsheet tool supporting the refinement of flowsheets has to assure such consistency constraints. To support this combination of complex refinement tasks with knowledge capture in the meta database implementation, we added a part-of abstraction to the conceptual object meta meta model, combined with the elementary distinction between process steps and streams (physically corresponding to chemical plant devices and connections between them) which is typical for the flowsheet. The resulting meta meta model (which is formally represented in logic) is graphically depicted in figure 6.

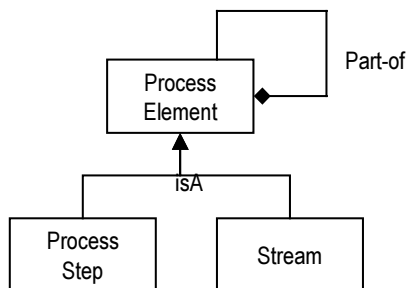


Fig. 6. Meta meta model for refinable flowsheet

3.3 Rich and Extensible Type System for Flowsheet Components

The organization of flowsheet components in an expressive type system is an indispensable prerequisite for guiding consistent flowsheet refinement. However, the richness and the rapid growth of knowledge goes beyond the abilities of typical object-oriented type systems : there are ten thousands of conceivable process step types and millions of possible stream types. Examples of refinement rules include:

- “if a process element has the same materials in its input and output streams, a feedback loop should be designed for them”;
- “if you add a reaction component, and the output of this reaction is a mixture of different materials, a separation unit should be added behind the reactor”;
- “if the boiling temperature of the ingredients within a mixture differ pairwise by more than 10°C, the separation is best implemented by a distillation column”.

The type system has to specify compatibility between flowsheet components along several dimensions (e.g. „U can be used as a *specialization* of V“ or „X can occur as part of a *decomposition* of Y“). It has to provide a semantically rich characterization of individual flowsheet components (e.g. the third rule above).

The type system must be extensible since knowledge about available flowsheet components and their properties is still rapidly evolving. Furthermore, the chemical engineer should be empowered to define his/her own flowsheet components and characterize them semantically in order to promote their reuse in future projects. User-defined flowsheet components emerge, e.g., from parametrizing generic flowsheet components or from aggregating complex components from simpler ones (e.g. by sequencing a set of distillation columns and heat exchangers).

Clearly, this richness cannot be hardcoded in the flowsheet editor. Instead, we encode declarative knowledge in the meta database of the process data warehouse. The data model of the flowsheet editor just knows about the most obvious and stable refinements of the meta meta model from figure 7, and about the existence of subtypes with different (user-definable) graphical symbols. The semantics of these subtypes, as well as rules and methods for their usage, is delegated via calls to the evolving amount of knowledge in the meta database of the process data warehouse, where it is accessed via querying and constraint analysis techniques. Considerable effort has been spent to find effective mechanisms for handling such large hierarchies in meta-data repositories based on deductive object-oriented [BaJa99] and/or description logic data models [Satt98]. The present implementation only covers some of these results.

4 Process Integration of the Flowsheet Editor

While the creation of wrappers for simple tool invocation/stoppage and basic data transfer is relatively simple, it only constitutes a rough approximation to the kind of process integration we envisioned in section 2. Fortunately, full process integration is only necessary for those tools in the chemical engineering process which have either complex tool-internal user interaction, or complex process interaction with several other tools. From the previous discussions, it should be obvious that the flowsheet editor is a prime example of such a tool. We therefore focused our efforts on process-integrating a flowsheet editor first, and to demonstrate process integration with other widely used tools (such as Excel spreadsheets) which are known from our practice surveys to interact a lot with the flowsheet.

4.1 Architecture and Basic Functionality

The development of a completely new flowsheet editor is a formidable task, and the result is unlikely to be able to compete with the rapid advances in graphical design tools. Our process-integrated flowsheet editor has therefore been realized according to the a-posteriori philosophy of IMPROVE, on top of an existing tool.

None of the commercially available flowsheet tools fulfills the requirements concerning flowsheet refinement and extensible type systems for flowsheet components. Hence, the existence of open interfaces, which allow the addition of the above-mentioned functionality (besides the process integration), became the most important criterion during the choice for a tool to be integrated. We finally selected VISIO [Visi98], a widely used tool for creating technical drawings. VISIO's strengths lie in its comprehensive and extensible symbol libraries and its add-on mechanisms based

on COM interfaces. These interfaces allow external extensions a fine-grained access to VISIO's internal object model.

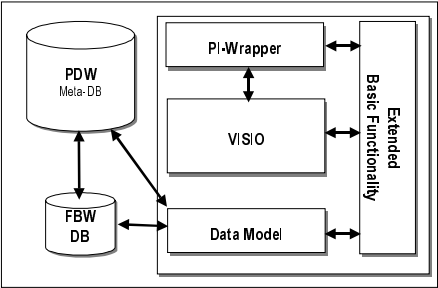


Fig. 7. Architecture of the Flowsheet Editor

Fig. 7 depicts the coarse-grained architecture of the flowsheet editor. Visio is the center of the architecture and provides the essential GUI-functionality for constructing flowsheets. The data model layer extends VISIO's object model by refinement structures and stores the flowsheet persistently in a (relational) DBMS. VISIO's GUI functionality and the data model layer are used by the extended basic functions which realize operations such as creating a new refinement level or navigating between different refinement levels. Figure 9 shows a refinement within the same drawing pane. This is done by expanding the right flowsheet node into a sub-drawing, thus automatically forcing the user to consider the balancing rules for inputs and outputs.

4.2 Wrapper for Process Integration

For integrating the flowsheet editor with the process model interpretation, we have adapted the PRIME approach [Poh*99]. Process integration in PRIME is based on explicit models of both the design processes and the tools to perform the processes. Process and tool models are integrated within the so-called environment model.

The interpretation of environment models enables the tools to adapt their behavior to the process definition and the current process state. A generic implementation framework defines the basic structure of a process-integrated tool and provides components for environment model interpretation and interaction with the process engine.

Originally meant for the development of *new* process-integrated tools, the implementation framework can also be used as a *wrapper* for *existing* tools. To be process-integratable, a tool has to offer a number of application programming interfaces (APIs): (1) *service invocation and feedback*: invoke individual services in a (running) tool instance and report the service results back to the process engine; (2) *command insertion*: enhance the user interface of the tool by new commands and bind them to externally defined process fragments; (3) *product display*: highlight currently relevant design objects in the user interface of the tool; (4) *selectability*: restrict the selection of product parts and activation of commands; (5) *selection notification*: notify the process engine about product and command selections in the tool.

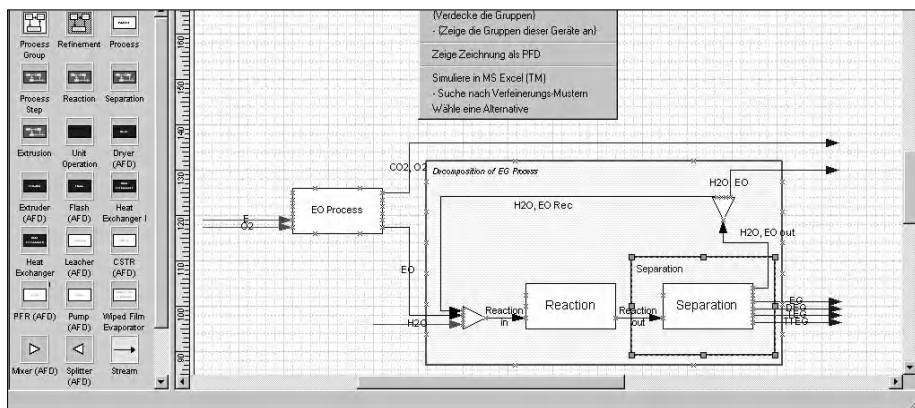


Fig. 8. Screenshot from flowsheet editor, demonstrating refinement on top of VISIO

The PRIME implementation framework uses these APIs when interpreting the relevant parts of the integrated tool and process models. It thereby brokers the interaction between the process engine and the tool. In our experiments, we achieved an almost complete process integration of VISIO. Only the process-sensitive restriction of tool functionality caused some problems since VISIO offers a large number of menu sets which have to be maintained by the PRIME-Framework.

4.3 Tool Data Model

The data model of the flowsheet editor has been designed in UML and implemented in C++ (Fig. 9). It is closely related to the partial models *Chemical Process* and *Plant* of the conceptual IMPROVE product data model. Its design was also influenced by emerging data exchange standards in chemical engineering (e.g. the process data exchange interface standard PDXI in the STEP context [DaGo95]).

The connectivity between flowsheet elements is defined at the level of abstract classes (*ProcessDevice*, *Port*, *Connector*, *Stream*), since it shares the same structure in both ADFs and PFDs. Flowsheet elements can be aggregated to *ProcessGroups*. *ProcessGroups* can be refined by other *ProcessGroups*. This enables the uniform definition of *n:m* relationships of different refinement types (decomposition, enrichment, specialization, realization) between flowsheet elements. As an essential prerequisite for extending the type system of flowsheet elements, only its abstract classes are directly defined in the data models (*Process*, *ProcessStep*, *Unit-Operation*, *ProcessEquipment*). As mentioned earlier, information about concrete flowsheet elements and their properties as well as compatibility constraints between flowsheet elements is maintained in the meta database of the process data warehouse which is consulted by the flowsheet editor at runtime.

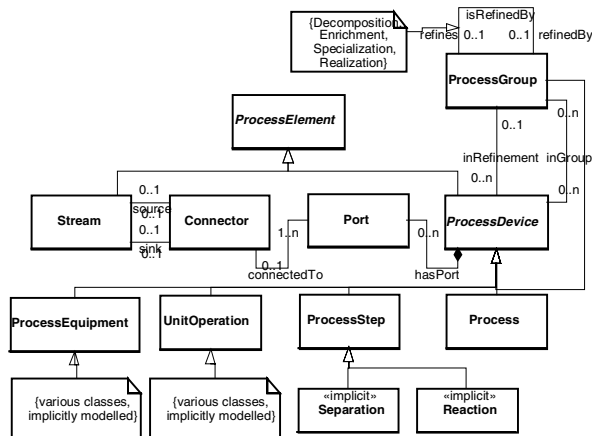


Fig. 9. Data model of the flowsheet editor

4.4 Example of a Tool-spanning Process Fragment

Fig. 10 shows the enactment of a process fragment involving multiple tools. In the VISIO-based flowsheet editor (left), the user has selected the refinement of a process group (step 1). The menu shows all tool actions as well as process fragments, which can be applied to the selected process group according to its type and the process definition. The user chooses the menu item *Simulation in MS Excel* (2). As this menu item is not related to an action provided by the flowsheet editor, a corresponding process fragment is enacted by the process engine (3). According to the definition of the process fragment, the process engine invokes Excel (right) and creates a spreadsheet which calculates the mass flow in the streams of the selected process group.

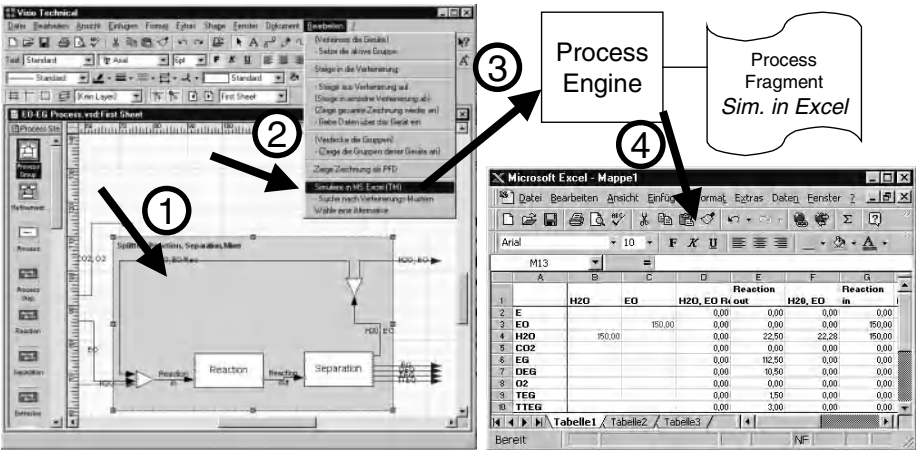


Fig. 10. Process fragment-guided tool integration

5 The Process Data Warehouse as Method Advisor

The method advisor functionality gives engineering tools such as the flowsheet editor access to conceptual knowledge about the chemical engineering domain stored in the metadata repository of the Process Data Warehouse.

To use the method advisor functionality, a tool has to ask the Process Data Warehouse for guidance in a specific situation given through an object and a method the tool wants to perform on it. The Process Data Warehouse now has to collect the necessary information to answer the request, i.e. further data found in the calling tool or in other tools and databases, e.g. the material database. After calculating the request based on the logical rules stored in the meta database, the Process Data Warehouse has to present the result in a suitable way. In the case of the flowsheet editor the request would consist of a flowsheet element selected by the user and a function like “refine” or “specialize”. As shown in section 3, additional information like the input and output streams of a process element, found in the flowsheet editor itself, or like the boiling temperature of the participant substances, found in the material database, could be needed. For the flowsheet editor a suitable result would be a set of process fragments, or one choice context combining several fragments.

Fig. 11 shows the method advisory part of the Process Data Warehouse in combination with the flowsheet editor and a material database. The meta database stores the partial models described in section 3, a set of queries defined on these partial models to answer the requests, and a set of meta-queries that represent the additional information needed to answer a specific query. The database trader stores what data can be retrieved from which tool or database and describes how to do that. To capture process context of knowledge reuse, a representation of the different process fragments known to the process engine is also contained in the data warehouse.

To collect the additional flowsheet elements and substance data we have developed CORBA wrappers for the participant tools. These wrappers are compliant with the standards defined in the CAPE-OPEN project.

We illustrate the interplay between the method advisor functionality of the process data warehouse and the enactment of a process fragment involving multiple tools by a short example. In Fig. 12 the chemical engineer has selected the process step „Glycole Separation“ (step 1) whose purpose it is to separate a mixture of different glycole types (EG², DEG³, TEG⁴, TTEG⁵) produced in a process step before. The chemical engineer wants to realize the „Glycole Separation“ by a set of separation apparatuses, but does not know which ones to use and how to connect them. Instead of using the generic, but unspecific refinement function provided by flowsheet editor itself, the chemical engineer invokes (via the interface of the flowsheet editor) the method advisor component of the Process Data Warehouse (2).

² Ethylene glycol

³ Diethylen glycol

⁴ Triethylene glycol

⁵ Tetraethylene glycol

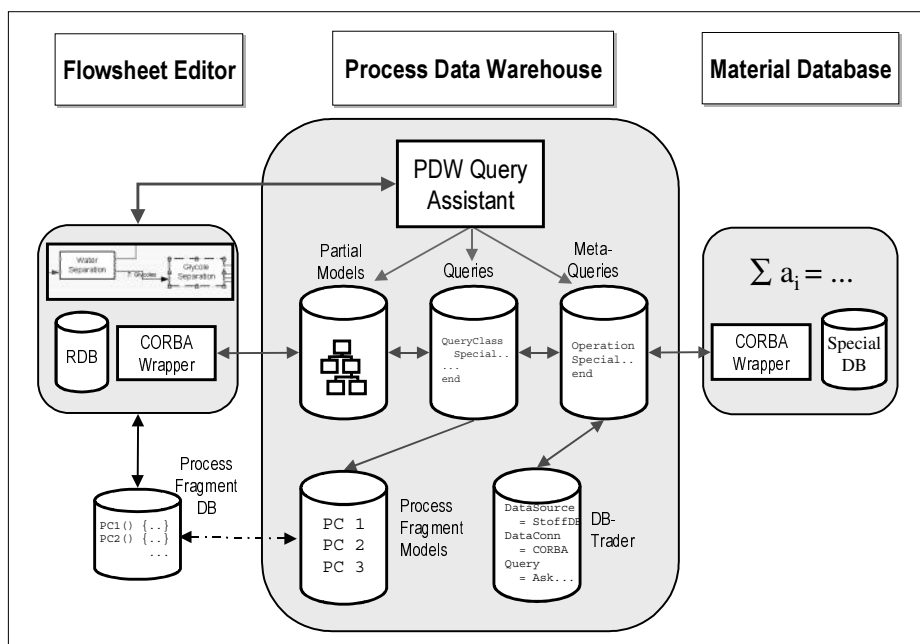


Fig. 11. Architecture of the method advisory function

For answering the query, the query assistant of the method advisor first retrieves all additional information required for analysing the current situation at hand from various data sources. In this case, the query assistant must first determine all input streams of the process step to be refined from the flowsheet database and then retrieve the relevant material data (here: boiling points) of the chemical components contained in the input streams from a material database (4). In our example, the current situation is evaluated to „Separation of a component mixture with sufficiently different boiling points“. The query assistant then retrieves possible process fragments which provide more specific procedural support for handling the current situation.

The resulting process fragment is activated by the process engine (5). According to the process fragment definition, the process engine suggests the refinement of the „Glycolic Separation“ by a sequence of three distillation columns (6). However, for determining the sequence in which the distillation columns are connected, it is important to assess the fraction of each component in the input stream. For this purpose, the process engine automatically creates an Excel spreadsheet which calculates the mass flow of each component in the input stream of the separation unit to be refined (7). According to the knowledge gained from this simulation, the chemical engineer can decide how to connect the different distillation columns. The resulting flowsheet is shown at the bottom of Fig. 12 (8).

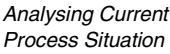


Fig. 12. Method Advisory enacting a tool spanning process fragment

6 Conclusion and Outlook

The chemical engineering domain brings out some problems in large-scale conceptual modeling which are perhaps less clear in the more widely discrete domains

such as mechanical or software engineering. In this paper, we focused on the question how to bring knowledge to the engineering workplace as directly as possible (i.e. in a process-integrated manner) even if this knowledge is complex and undergoes continuous change. Our solution to this problem was the combination of a framework for process integration of engineering tools, with a knowledge-based repository. Simplified versions of the data model and flowsheet editor reported here are being implemented for production usage in the company where we conducted the initial requirements workshop. From their usage, we are expecting experimental data concerning the actual potential of our approach.

The metadata provided by the process data warehouse also allow the control of data exchange and transformation between different tools, and the targeted capture and visualization of trace data as a basis for creating the kind of method knowledge which has been used in our examples. This explicit handling of traces is the major focus of our present work; it requires, in particular, the design of a process query language in order to selectively visualize and abstract traces captured by the PRIME environment across multiple tools [DöPo98].

Moreover, there is an additional complication in process data warehouses. An empirical study of requirements traceability in the American software industries [Rame98] showed that process capture is not limited to conceptual objects. Equally important is the representation of how this conceptual knowledge is grouped in documents, as documents are the unit of management control in engineering processes; this is further augmented by the capture of stakeholder roles, both with respect to content (the stakeholder as a source of requirements or responsible for a content) and documents (the stakeholder as owner, reader, ... of documents). The expanded meta meta model of the process data warehouse we aim to support is shown in fig. 13. This figure also shows how this meta meta model will be used to link the contributions of projects B1-B4 from figure 1 within the overall IMPROVE effort.

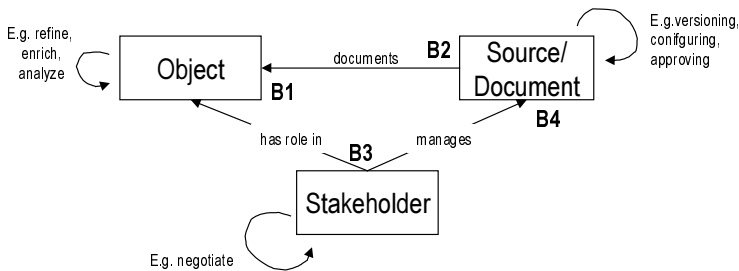


Fig. 13. Meta meta model of process data warehouse

References

[BaJa99] Baumeister, M., Jarke, M.: Compaction of large class hierarchies in databases for chemical engineering. Proc. BTW 99 (Freiburg, Germany), Springer-Verlag 1999, 343-361.

- [BaSM99] Bayer, B., Schneider, R. und Marquardt, W.: Product Data Modeling for Chemical Process Design. Proc. European Concurrent Engineering Conference, Erlangen, Germany, April 1999.
- [Blas97] Blass, E.: Entwicklung verfahrenstechnischer Prozesse - Methoden, Zielsuche, Lösungssuche, Lösungsauswahl. Springer Verlag Berlin Heidelberg, 1997.
- [Bra*99] Braunschweig, Jarke, M., Köller, J., Marquardt, W., v.Wedel, L.: CAPE-OPEN – experiences from a standardization effort in chemical industries. Proc. Intl. Conf. Standardization and Integration in Information Technology (SIIT 99), Aachen, September 1999.
- [CDL*98] Calvanese, D., deGiacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Information integration – conceptual modeling and reasoning support. Proc. CoopIS 98, New York 1998, 280-291.
- [DaGo95] Dalton, C.M., Goldfarb, S.: PDXI, a Progress Report. In: Proc. CHEMPUTERS Europe II Conference, Noordwijk, Niederlande, Oct. 1995.
- [Döm*96] Dömgies, R., Pohl, K., Jarke, M., Lohmann, B. und Marquardt, W.: PRO-ART/CE - An Environment for Managing Chemical Process Simulation Models. In: Proc. 10th European Simulation Multiconference, Budapest, Hungary, 1996, S. 1012-1016.
- [DöPo98] Dömgies, R., Pohl, K.: Adapting traceability environments to project-specific needs. Comm. ACM 41, 12 (1998), 54-62.
- [Doug88] Douglas, J.: Conceptual Design of Chemical Processes. McGraw-Hill, 1988.
- [JGJ*95] Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., and Ehrer, S.: ConceptBase-a deductive object base for meta data management. J. Intelligent Information Systems, 4(2):167-192, 1995.
- [JJQV89] Jarke, M., Jeusfeld, M.A., Quix, C., Vassiladis, P.: Architecture and quality of data warehouses. Proc. CAISE 98, Pisa, Italy.
- [JaMa96] Jarke, M. und Marquardt, W.: Design and Evaluation of Computer-Aided Process-Modeling Tools. Proc. Intl. Symp. Intelligent Systems in Process Engineering (ISPE 95), Snowmass, Co, 1995, AIChE Press 1996..
- [Jar*99] Jarke, M., Tresp, Ch., Becks, A., Köller, J. und Braunschweig, B.: Designing Standards for Open Simulation Environments in the Chemical Industries: A Computer-Supported Use-Case Approach. Proc. 4th Intl. Conf. Systems Engineering (INCOSE 99), Brighton, UK, 1999.
- [LeSK95] Levy, A., Srivastava, D., Kirkk, T.: Data model and query evaluation in global information systems. J. Intelligent Information Systems 5, 2 (1995), 121-143.
- [Lohm98] Lohmann, B.: Verfahrenstechnische Modellierungsabläufe. Dissertation, RWTH Aachen, VDI Verlag Düsseldorf, Fortschritts-Berichte VDI, Reihe 3, Nr. 531, 1998.
- [MaGG93] Marquardt, W., Gerstlauer, A. und Gilles, E.D.: Modeling and Representation of Complex Objects: A Chemical Engineering Perspective. In: Proc. 6th Intl. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Edinburgh, Scotland, 1993, S. 219-228.
- [McJo88] McGuire, M.L. und Jones, J.K.: Maximizing the Potential of Process Engineering Databases. In: AIChE Annual Technical Meeting, Houston, TX, USA, 1988.
- [NaWe99] Nagl, M. und Westfechtel, B. (Hrsg.): Integration von Entwicklungssystemen in Ingenieur Anwendungen. Springer-Verlag 1999.
- [Poh*99] Pohl, K., Weidenhaupt, K., Dömgies, R., Haumer, P., Jarke, M., Klamma, R.: PRIME: Towards Process-Integrated Environments. To appear in: ACM Transactions on Software Engineering and Methodology.
- [Rame98] Ramesh, B.: Factors influencing requirements traceability practice. Comm. ACM 41, 12 (1998), 37-44.
- [Satt98] Sattler, U.: Terminological Knowledge Representation Systems in a Process Engineering Application. Dissertation, RWTH Aachen, 1998.
- [Visi98] Visio Corp: Developing VISIO solutions, version 5.0. Seattle, Wash., 1998.

ER Model, XML and the Web

P.P.S. Chen

Louisiana State University, USA

The ER Model and its variants have been used successfully in data modeling and database design in more than twenty years. In the past few years, the Web has become an increasingly popular user interface to files and databases. The XML, which is developed by the World Wide Web Consortium, is positioned to be the main stream markup language for the web of the future. Recently, several XML working groups are in the process of developing specifications related to data types, schemas, and data models. Whether the ER model (or its variants) can serve as the model of the web is a subject of a debate within the XML working groups. In this talk, we will look at some of the current modules of XML such as DTD, RDF, XLink, XPointer, and XL Schema. We will then show the similarities and differences between the main concepts in these modules and the main concepts in the ER model. Then, we will present the reasons why the ER model is a good candidate for the model of the web.

Author Index

- Z. Abdelouahab, 31
D. Agrawal, 161
A. Analyti, 475
A. Artale, 81
- M. Balaban, 369
S. Balko, 445
B. Benatallah, 16
- J.H. Canós, 278
S. Castano, 146
D. Castelli, 1
S.S.-M. Chan, 47
P.P.S. Chen, 538
A.E.M. Ciarlini, 460
S. Conrad, 413,445
P. Constantopoulos, 475
D. Costal, 62
- V. De Antonellis, 146
J.L.G. Dietz, 188
K.R. Dittrich, 263
J. Dullea, 384
- J. Eder, 430
A. El Abbadi, 161
- A. Flitman, 114
E. Franconi, 81
H. Frank, 430
A.L. Furtado, 460
- M.M. Gammoudi, 31
G. Gardarin, 506
A. Geppert, 263
- T.-D. Han, 203
M. Heisel, 309
E. Hildebrandt, 445
M. Höding, 445
- M. Jarke, 520
L.A. Kalinichenko, 131
Z. Kedad, 325
M. Kradolfer, 263
- Z. Lacroix, 176
N. Lammari, 218
M.L. Lee, 131
S.Y. Lee, 131
J. Lewerenz, 354
Q. Li, 47
T.W. Ling, 131
T. List, 520
J. Lonchamp, 233
B. F. Loscio, 293
- P. McBrien, 96
E. Métais, 325
D.L. Moody, 114
- I. Nafkha, 31
W. Ng, 399
T.D. Ngoc, 506
- A. Olivé, 62
- O. Pastor, 278
A. Poulouvassilis, 96
G. Preuner, 413
S. Purao, 203
- I. Ramos, 278
- G. Saake, 445
M.-R. Sancho, 62
K.-D. Schewe, 354
J.W. Schmidt, 248
I. Schmitt, 445
R. Schuette, 490
K. Schwartz, 445
H.-W. Sehring, 248

F. Sha, 506

P. Shoval, 369

I.-Y. Song, 384

J. Souquières, 309

N. Spyratos, 475

I. Stanoi, 161

V.C. Storey, 203

B. Thalheim, 354

M. Theodorakis, 475

D. Theodoratos, 340

C. Türker, 445

V.M.P. Vidal, 293

K. Weidenhaupt, 520